STORM: A Framework for Integrated Routing, Scheduling, and Traffic Management in Ad Hoc Networks

J.J. Garcia-Luna-Aceves, *Fellow*, *IEEE*, and Rolando Menchaca-Mendez

Abstract—A cross-layer framework is introduced for the effective dissemination of real-time and elastic traffic in multihop wireless networks called Scheduling and Traffic Management in Ordered Routing Meshes (STORM). Unicast and multicast routes are established in coordination with the scheduling of transmissions and bandwidth reservations in a way that bandwidth and delay guarantees can be enforced on a per-hop and end-to-end basis. The routes established in STORM are shown to be loop-free and real-time packets forwarded along these routes are shown to have bounded end-to-end delays. Results from detailed simulation experiments show that, compared to a protocol stack consisting of 802.11 DCF for channel access, AODV or OLSR for unicast routing, and ODMRP for multicast routing, STORM attains similar or better performance for elastic traffic, and up to two orders of magnitude improvement in end-to-end delays, with twice the amount of data delivery for real-time traffic while inducing considerably less communication overhead.

Index Terms—Cross-layer design, integrated routing, channel access, traffic management.

1 Introduction

The price, performance, and form factors of processors, radios and storage elements are such that wireless ad hoc networks have become a reality. In theory, they constitute an ideal vehicle to support disaster-relief and battlefield operations, emergency search and rescue missions, and many other distributed applications on the move. However, in practice, these applications cannot be supported effectively in these networks today. To a large extent, this is due to the fact that the protocol architectures used in wireless ad hoc networks, and mobile ad hoc networks (MANET) in particular, are derivatives of the protocol-stack architectures developed for wired networks and the Internet. The stark differences between wireless and wired networks call for a cross-layer approach to managing network resources in support of information dissemination. Section 2 provides a small sample of the considerable prior work focusing on cross-layer approaches to routing and transmission scheduling. The solutions that have been proposed in the past are either based on centralized algorithms requiring too much information at each node, or do not integrate routing and scheduling with the establishment of bandwidth reservations and traffic management. Furthermore, prior solutions do not address the integration of unicast and multicast routing with transmission scheduling.

The main contribution of this paper is to introduce a new cross-layer framework for the dissemination of unicast and multicast flows that may be real time (e.g., voice conversations) or elastic (e.g., http). Section 3 presents an overview of Scheduling and Traffic Management in Ordered Routing Meshes (STORM). Section 4 describes the channel structure assumed in STORM and the priority-based queuing system used to handle signaling traffic, elastic data flows and real-time flows. Sections 5 and 6 describe the neighbor protocol and distributed transmission scheduling used in STORM. This scheduling is coupled with a transaction-oriented end-to-end reservation scheme for time slots, which is driven by the establishment of routes to provide end-to-end bandwidth and delay guarantees. Section 7 presents the interest-driven approach used in STORM for unicast and multicast routing. STORM is the first cross-layering scheme that provides flow-ordered routing meshes consisting of multiple paths from sources to destinations over which relays are capable of establishing channel reservations that meet the end-to-end requirements of the flows being routed. In addition, STORM confines the majority of the signaling needed to maintain such meshes to those nodes in the mesh and neighbors of those nodes.

Section 8 shows that STORM establishes and maintains loop-free routes from sources to destinations and, just as important, that the reservations established along these paths provide bounded end-to-end delays. Section 9 describes the results of detailed simulation experiments used to study the performance of STORM and compare it with the performance of a traditional MANET protocol stack, which consists of the IEEE 802.11 DCF [9] for channel access working independently of the routing protocols used for unicasting (AODV [18], OLSR [12]) and multicasting

- *J.J. Garcia-Luna-Aceves is with the Department of Computer Engineering, Jack Baskin School of Engineering, University of California, 317 Engineering 2 Bldg, Santa Cruz, CA 95064, and the Palo Alto Research Center, Palo Alto, CA 94304. E-mail: jj@soe.ucsc.edu.*
- *R. Menchaca-Mendez is with the Computer Research Center of the Mexican National Polytechnic Institute, Av. Juan de Dios Bátiz s/n, Esq. Miguel Othón de Mendizabal, Col. Nueva Industrial Vallejo, C.P 07738, Mexico D.F. E-mail: rmen@cic.ipn.mx.*

Manuscript received 17 Aug. 2010; revised 1 July 2011; accepted 15 July 2011; published online 30 July 2011.
For information on obtaining reprints of this article, please send e-mail to: tmc@computer.org, and reference IEEECS Log Number TMC-2010-08-0388. Digital Object Identifier no. 10.1109/TMC.2011.157.

(ODMRP [13]). We present results for multicast traffic only and for combined unicast and multicast traffic. The results show that STORM outperforms the traditional approach for elastic traffic, and that it attains end-to-end delays that comply with the ITU recommendations for voice applications [11], even in the presence of a highly loaded network. The simulation experiments consider all the control and data traffic generated by each protocol stack and show that STORM incurs half the total overhead (control packets and redundant data packets) while rendering up to two orders of magnitude improvement in end-to-end delays compared to traditional routing approaches over 802.11. STORM is best suited for ad hoc networks subject to a variety of multicast and unicast traffic, with some being subject to delay or bandwidth constraints; however, STORM can provide some performance improvements even in those cases where the traffic is unicast and elastic (e.g., http), because it avoids most packet collisions and limits the signaling overhead needed for routing. The prize that must be paid for this improved performance is the needed cross layering of routing with scheduling and queuing, which calls for a redesign of the protocol stack, and the need to enforce clock synchronization to establish time-slotted channel access.

2 Related Work

Due to space limitations we present only a small representative sample of prior cross-layering approaches attempting to make routing and channel access more efficient in ad hoc networks. Chen and Heinzelman [5] provide a comprehensive survey on routing protocols that provide some sort of support for QoS in MANETs, and Melodia et al. [15] present a survey of cross-layer protocols for wireless sensor networks.

Multiple access collision avoidance with piggyback reservations (MACA/PR) [14] was one of the first approaches attempting to integrate channel access, routing, and traffic management. MACA/PR, which supports only unicast traffic, extends the IEEE 802.11 DCF to incorporate a bandwidth reservation mechanism and includes a modified version of DSDV [17] that keeps track of the bandwidth of the shortest paths to each destination and the maximum bandwidth available over all possible paths. The first data packet of a real-time flow makes reservations along the path for subsequent packets in the connection. One-hop scheduling information is piggybacked in data packets and ACKs which reserve time-synchronized windows at specified time intervals. Reservations are made taking into account only two-hop neighborhood information and without coordination with the routing protocol.

DARE [4] is a channel access protocol for MANETs that provides end-to-end reservations. It is based on request-to-reserve messages that travel from sources to destinations through routes established by a traditional on-demand routing protocol. Destinations reply with clear-to-reserve messages that travel along reverse paths establishing the actual reservations. Data packets also contain reservation information and are used to refresh the reservations tables. The main limitations of DARE are that reservations are established at each hop of a path independently of the other hops in the path, and routing decisions do not consider information regarding reservations or any other data collected for channel access. Cai et al. [3] propose an algorithm for end-to-end bandwidth allocation that focuses on maximizing the number of flows with bandwidth restrictions that a MANET can accommodate. The disadvantage of this algorithm is that it requires global resource information along the route and the route itself is not considered in the optimization algorithm. Setton et al. [20] propose a cross-layer framework that incorporates adaptations across all layers of the protocol stack. The proposed framework, however, is mostly based on centralized algorithms and a link-state approach is needed, which is not well suited for the highly dynamic MANETs or very large ad hoc networks.

In the context of multicast communication, most of the work has focused on static networks (i.e., [21], [22]). In these works, the authors formulate the joint multicast routing and power control problem [22] or the network planning problem [21] as a cross-layer optimization problem. However, no proposals have been made on the integration of scheduling and routing for many-to-many communication using distributed algorithms based on local information.

3 STORM

STORM assumes that nodes share a single wireless channel organized into time frames consisting of a fixed number of time slots. The objective in STORM is to orchestrate the scheduling, routing, and traffic management functions of a multihop wireless network in a way that sources and destinations of flows perceive the network as a virtual link dedicated to the dissemination of those flows.

Accessing the time slots of each frame is based on a combination of distributed elections of available time slots and reservations of time slots. For those time slots that have not been reserved, nodes use a distributed election algorithm based on hashing functions of node identifiers. A virtual link is created to support an individual real-time data flow and is implemented by a set of nodes located at directed meshes connecting sources to destinations. These meshes are computed by means of an interest-driven [16] routing algorithm that establishes an ordering over the nodes based on their distances to the destination and the bandwidth available to them. To provide the abstraction of a virtual link, the routing algorithm also computes an end-to-end channel access schedule for each data flow. The schedules generated by STORM are such that delay guarantees can be enforced on a per-hop and end-to-end basis. The end-to-end schedules are instantiated by the reservation protocol when the first data packet traverses the flow's routing mesh.

The routing meshes established by STORM provide a fast and efficient way of repairing routes, because they contain extra paths that can be used in case of link breaks. This reduces the impact of node mobility on the quality of service perceived by real-time flows. In addition, the routing algorithm establishes *enclaves*, which restrict the dissemination of control information to those nodes that are likely to participate as forwarders of a given data flow, rather than the entire network.

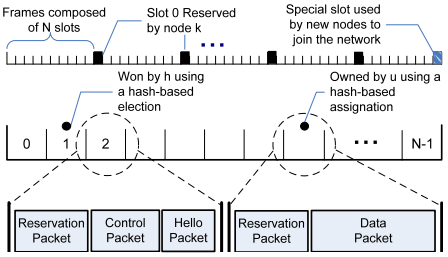

Fig. 1. Channel structure in STORM.

STORM uses reservations and a priority-based queuing system to implement and preserve the per-flow channel access schedules. Nodes reserve time slots on behalf of real-time data flows according to their end-to-end schedules and use a priority-based queuing system to select the packets that are transmitted on each slot. The queuing system is composed of queues for signaling traffic, elastic (non real-time) traffic and real-time traffic. A real-time queue is created for every new real-time flow traversing a node, and it is associated with the time slots reserved for the flow. During the time slot reserved for a given flow, its associated queue is given a higher priority than those assigned to other data queues. This way, STORM establishes a dedicated queuing network for each real-time flow to avoid interference among multiple real-time flows traversing the same nodes.

4 Channel Structure and Traffic Management

Nodes share the same frequency band, and we assume that clock synchronization among the nodes in the network is achieved through a multihop time synchronization scheme such as the one implemented in Soft-TDMAC [6] which is a TDMA-based MAC protocol that runs over commodity 802.11 hardware. Nodes access the common channel assuming that it is organized using a time-division multiple access structure, which we call STORM frame and is illustrated in Fig. 1. Each STORM frame is composed of N time slots (from slot 0 to slot $N-1$) and we use the position of a slot within the STORM frame as the identifier of the slot. A STORM frame does not have any particular structure and any time slot can be used to transmit a sequence of packets (signaling or data). There is only one special purpose time slot used to admit new nodes to the network. These admission time slots occur every A time slots, with $A \gg N$, and are used by nodes to transmit their first hello packets on a contention basis.

When a node is allowed to transmit over a time slot, it fits as many packets as possible in it. Packets are selected from the local transmission queues, which are FIFO and are served using a priority-based algorithm.

Reservation packets have the highest priority (p_{Rsv}), because quick consensus is needed on which nodes should have access to which time slots. The next priority is given to network-layer signaling packets (p_{ctr}), and data packets waiting in data queues have the lowest priority. Data queues can be either elastic or real-time, and real-time queues are assigned higher priority (p_{RT}) than the priority given to elastic queues ($p_{elastic}$), given that jitter and latencies are not as important for the latter. The priority of a real-time queue created for flow f is increased from p_{RT} to p_{RT+} if the current time slot t was reserved on behalf of flow f. Hello packets are transmitted with the lowest priority ($p_{Hello-} < p_{elastic}$) if more than $hello\_period/2$ seconds but fewer than $hello\_period$ seconds have elapsed since the last time a hello packet was transmitted, because there is no need for the information yet. However, if more than $hello\_period$ seconds have elapsed, then the neighborhood information must be refreshed and hence the priority of the hello packet is set to $p_{Hello+} > p_{ctr}$.

To summarize, during a time slot allocated to a node, the relationships among traffic priorities are: $p_{Hello-} < p_{elastic} < p_{RT} < p_{RT+} < p_{ctr} < p_{Hello+} < p_{Rsv}$.

5 Neighbor Protocol

Routing, reservations, and transmission scheduling in STORM use distributed algorithms that require each node to know the nodes within its two-hop neighborhood. The neighborhood of a node consists of those nodes whose transmissions the node can decode, which we call one-hop neighbors, and the one-hop neighbors of those nodes are called two-hop neighbors. More formally, let $G = (V, E)$ be an undirected graph with a set of vertices V representing the set of nodes present in a wireless ad hoc network and a set of edges E. Any two nodes u and v share an edge $(u, v) \in E$ if they are one-hop neighbors (i.e., within radio transmission range) of each other. For any node $u \in V$, we denote $\mathbb{N}(u) = \{v : (u, v) \in E\}$ as the one-hop neighborhood of u and $\mathbb{N}(\mathbb{N}(u))$ as the two-hop neighborhood of u.

To gather two-hop neighborhood information, each node transmits *hello* messages periodically every $hello\_period$ seconds, and each such message contains a list of tuples for the node itself and for each of its one-hop neighbors. Each tuple is composed of a node identifier, a list of the identifiers of the time slots reserved by the node, and the length of the list of reserved slots.

Each node stores the last hello message received from each one-hop neighbor (or simply neighbor) in its *neighbor list*. A neighbor is deleted from the neighbor list if no hello message is received from that neighbor in three consecutive hello periods.

It is worth noting that the neighbor protocol in STORM is very similar to approaches used in traditional routing protocols that also require neighborhood information (e.g., OLSR) in that hello messages are transmitted unreliably but persistently, and convey information about local neighborhoods.

The neighbor protocol is also used to detect when two nodes in a two-hop neighborhood have reserved the same slot. To resolve a conflicting reservation, the node with the larger identifier keeps its reservation over the particular slot, whereas the node with the lower identifier has to give up its current reservation and start a new reservation transaction over a different slot. The main source of these conflicting reservations is node mobility, which changes the neighborhood of nodes. The neighborhood information contained in hello messages allows nodes to detect these collisions before the conflicting nodes become one-hop neighbors.

6 Transmission Scheduling

The channel access algorithm used in STORM (see Algorithm 1) consists of three simple ways to determine which node should transmit in a time slot. On every slot t with identifier $(t \bmod N)$, node u with identifier id^u first checks if it is the owner of the slot (i.e., if $(t + id^u) \bmod N = 0$) and if so, u can access the channel. If node u does not own the slot, it checks if the owner is present in its two-hop neighborhood (i.e., if there is $v \in \mathbb{N}(\mathbb{N}(u))$ such that $(t + id^v) \bmod N = 0$). If this is the case, then node u listens to the channel.

Algorithm 1: ChannelAccess($\mathbb{N}(\mathbb{N}(u), u, t)$

```
if (t + id^u) mod N = 0 then
    C ← {v : v ∈ N(N(u)) ∧ (t + id^v) mod N = 0};
    if C = ∅ then
        Access channel;
    else
        compute p_u^t = Hash(id^u ⊕ t) ⊕ id^u;
        ∀v ∈ C compute p_v^t = Hash(id^v ⊕ t) ⊕ id^v;
        if ∀v ∈ C : p_u^t > p_v^t then
            Access channel;
        else
            Listen to channel;
else
    if ∃v ∈ N(N(u)) | (t + id^v) mod N = 0 then
        Listen to channel;
    else
        if u has reservation on t mod N then
            Access channel;
        else
            if ∃v ∈ N(N(u)) | v has reservation on t mod N then
                Listen to channel;
            else
                for v ∈ N(N(u)) ∪ {u} do
                    compute p_v^t = Hash(id^v ⊕ t) ⊕ id^v;
                if ∀v ∈ N(N(u)) | p_u^t > p_v^t then
                    Access channel;
                else
                    Listen to channel;
```

If there is a collision of the mod operation, the time slot is not considered as owned by any of the contending nodes and a hash-based election is held among the nodes participating in the collision. If the owner of the time slot is not present in the two-hop neighborhood, node u checks if it has a reservation on the slot $(t \bmod N)$, in which case it can access the channel. Otherwise, node u checks if there is $v \in \mathbb{N}(\mathbb{N}(u))$ such that v has reserved the slot $(t \bmod N)$. If that node exists, node u listens to the channel.

If none of the two previous conditions are met, node u employs a hash-based election scheme [1] to select the node that can access the network. Node u computes the priority of each node v in its two-hop neighborhood as $p_v^t = Hash(id^v \oplus t) \oplus id^v$ where $\oplus$ is the concatenation operator and $Hash$ is any hash function with good uniformity properties. Node u can access the channel if $p_u^t > p_v^t$ for any node v in its two-hop neighborhood. Otherwise, node u listens to the channel.

The rationale for this approach is twofold. First it attains the simplicity and efficiency of prior schemes based on hash-based elections (e.g., NAMA [1] and ROMA [2]), which have been shown via analysis, simulations, and real deployments to establish dynamic transmission schedules in real dynamic networks, some with vehicular speeds, and to provide higher throughput than 802.11 DCF. Second, it alleviates the "coupon collector's" problem [7] that arises in probabilistic hash-based elections, which is important to establish and modify schedules quickly for real-time traffic. Fig. 2 illustrates some of the benefits of our scheme by showing the results of a simulation experiment that compares the average number of slots needed in NAMA and STORM (even without reservations) for all the nodes in a two-hop neighborhood to access the channel. In the experiment, the size of the two-hop neighborhood was varied from 30 to 80 nodes with steps of five nodes. Each point in the graphs represents the average over 100,000 independent runs. The results indicate that the use of a simple slot assignation technique such as slot *ownership* reduces the mean and standard deviation of the channel access delay per node. Further work is also needed to solve the coupon collector's problem more effectively.

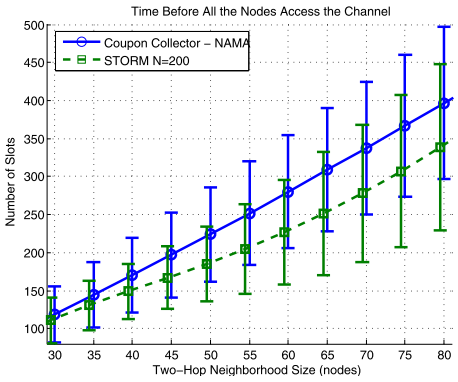

Fig. 2. NAMA versus STORM.

6.1 Channel Reservations

When a node becomes part of a persistent real-time data flow, i.e., when it starts transmitting real-time data packets for a source-destination pair, it uses the reservation protocol to reserve future slots to be used on behalf of that particular real-time flow. Unlike many prior channel access schemes, the selection of a particular slot $r \in [0, N-1]$ in the STORM frame is controlled by the routing layer, so that the channel access schedules of the relays of the flow are *flow ordered*. The relays of a flow are *flow ordered* if every single one of them can access the channel in a time-ordered sequence of slots. To achieve this behavior, a node x that is relaying data packets toward destination D employs (1) to compute the interval of flow-ordered slot identifiers corresponding to its current distance to D (d_D^x). In (1), $slot_D$ is a reference selected at random by D from the interval $[0, N-1]$ and $\delta \in \{1, -1\}$ indicates the direction with which data packets are flowing at node x. For the case of unicast flows δ always takes the value of 1, whereas for multicast flows it takes the value of 1 if the data packets are traveling from the source toward the core of the group, and -1 if the data packets are traveling from the core to the receivers. The objective of δ in (1) is to establish two consecutive schedules, one from the source to the core using $\delta = 1$ and the other from the core to the receivers using $\delta = -1$ (see Fig. 3)

$$[(slot_D - \delta d_D^x \Delta) \bmod N, ((slot_D - \delta d_D^x \Delta) \bmod N + \Delta) \bmod N). \quad (1)$$

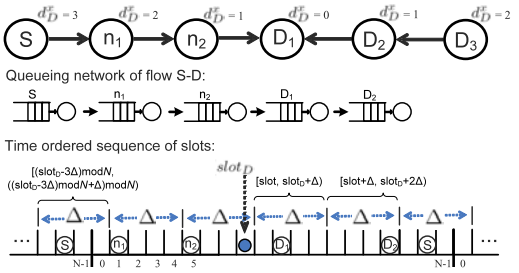

Fig. 3. End-to-end reservations.

By reserving slots from these intervals, any two consecutive nodes n_i and n_{i+1} (with distances to destination D: $d_D^{n_i} + 1 = d_D^{n_{i+1}}$) that lay in a successor path $p = \{s, n_1, n_2, \ldots, D\}$ established by the routing algorithm will have the right to access the channel within a maximum time of $2\Delta\tau$ seconds, where Δ is the size in number of slots of the interval defined by (1) and τ is the length in seconds of each time slot. Values of Δ close to the length of the STORM frame (N) have the effect of imposing no ordering when establishing the reservations, while values of Δ close to one impose very strong restrictions when looking for a suitable free time slot. Fig. 3 shows an example where a data flow between source S and destination D composed of nodes D_1, D_2, and D_3 is supported by nodes n_1 and n_2. In the figure, nodes n_1, n_2, D_1, D_2, and S use (1), a reference slot $slot_D$ disseminated by the core D_1 of the destination and their distances to D_1 to compute their corresponding intervals of flow ordered slot identifiers. The interval computed by n_1 equals $[slot_D - 2\Delta, slot_D - \Delta)$ because it is located two hops away from D_1 and data packets are flowing from the source to the core ($\delta = 1$), whereas the interval computed by D_2 equals $[slot_D + \Delta, slot_D + 2\Delta)$ because it has a one distance to the core and data packets are flowing from the core to a receiver ($\delta = -1$). The figure also illustrates how the mod operations used in (1) are needed to "wrap around" the STORM frame, such as in the case of node S.

In addition to the reservation of slots, a node that becomes part of a real-time flow has to create a new real-time queue associated with the flow and with the set of slots reserved on behalf of that flow. As we mentioned in Section 4, during a reserved time slot, the priority of this particular queue is raised to p_{RT+}, which is higher than the priority of the remaining data queues (real-time or elastic). Hence, no packet from a different flow is transmitted during these slots unless the queue under consideration is empty. This way, STORM guarantees that the bandwidth reserved on behalf of a real-time flow is actually used by that data flow, and that different flows that traverse the same node do not interfere with each other breaking the per-flow ordering of the end-to-end reservations.

As it is described in Section 7, the successor paths employed to transport real-time flows are established using nodes that have reported to have available slots in their intervals of flow ordered slot identifiers. This way, STORM increments the probability that these nodes successfully reserve the adequate slots in the STORM frame.

6.1.1 Reservations: Control Signaling

The process of reserving a time slot begins by finding a free slot in the interval defined by (1). A **free slot** for node x is a slot not currently owned or reserved by any node in its two-hop neighborhood, including x itself, and that is not in the process of being reserved. The information needed to verify these conditions is stored in three data structures, namely, the *Neighbor Lists*, the *Ongoing Reservation Lists*, and the *Reserved Slot List*. The Neighbor Lists stores information regarding the identity of nodes and reserved slots within the two-hop neighborhood and is maintained by the Neighbor Protocol. The Ongoing Reservation Lists stores information regarding ongoing reservations and is maintained by the Reservation Protocol. The Reserved Slot List keeps track of the slots that are currently reserved by the local node x. For each reserved slot, this list also stores the identifier of the queue that is associated with the slot. This queue's identifier is further used to update the priority of that particular queue (from p_{RT} to p_{RT+}) during the reserved slot.

Once a free slot $slot_c$ is identified, a node with identifier id^x transmits a Reservation Request $RsR^x = (id^x, slot_c)$ packet to its neighbors and waits for $N\tau$ seconds to collect the replies from them. If all the nodes $v \in \mathbb{N}(x)$ reply with Reservation Granted $RsG = (id^x, slot_c, id^v)$ packets granting the reservation, and neither $RsR^y = (id^y, slot_c)$ such that $id^y > id^x$, nor $RsG = (id^y, slot_c, id^w)$ such that $id^y > id^x$ are received; then node x considers the slot $slot_c$ as reserved by itself and moves that slot from its list of ongoing reservations to its list of reserved slots. Otherwise, node x selects the next free slot in the interval and transmits a new reservation request. This procedure is repeated until a time slot is successfully reserved or all the free slots in the interval defined by (1) have been tested. In the latter case, nodes wait for a mesh announcement (MA) period (e.g., 3 seconds) and retry the reservation process.

Upon reception of an $RsR^x = (id^x, slot_c)$, a node with identifier id^u replies with an $RsG = (id^x, slot_c, id^u)$ granting the reservation if none of the following conditions are true: the owner of slot $slot_c$ is present in the one-hop neighborhood of node u ($\nexists v \in \mathbb{N}(u) : (slot_c + id^u) \bmod N = 0$), a node in u's one-hop neighborhood holds a reservation over $slot_c$, or there is a concurrent request owned by a node with identifier id^y such that $id^y > id^x$. Otherwise, node u replies with a Reservation Denied $RsD = (id^x, slot_c, id^u)$ packet.

Nodes refresh their reservations by means of hello messages for as long as they are part of the corresponding flows. When a node receives a hello message from the owner of a given reservation that no longer contains that reservation, it considers that reservation as terminated.

6.1.2 Maintenance of End-to-End Reservations

During the lifespan of a real-time flow, nodes must ensure that the end-to-end sequence of reservations is maintained, even in the presence of topological changes. If node x is actively relaying real-time data packets toward a given destination and detects that its distance to that destination has changed from d_D^x to $d_D'^x$, then x cancels all its current reservations related to that destination and requests a new set of reservations on the appropriate slots, i.e., slots in

$$[(slot_D - \delta d_D'^x \Delta) \bmod N, ((slot_D - \delta d_D'^x \Delta) \bmod N + \Delta) \bmod N).$$

Similarly, if the value of δ of a given flow changes from δ to δ', then x cancels all its current reservations related to that

flow and requests a new set of reservations on the appropriate slots, i.e., slots in $[(slot_D - \delta' d_D^x \Delta) \bmod N, ((slot_D - \delta' d_D^x \Delta) \bmod N + \Delta) \bmod N)$. In our implementation, nodes do not immediately react to a change in their value of δ or their distances to the destination; instead, they enter a hysteresis period to verify that the change is stable and not a transient event.

7 Loop-Free Interest-Driven Routing and End-to-End Scheduling

7.1 Meshes and Enclaves

Routing in STORM is based on *destination meshes*, *routing meshes*, and *enclaves*. To integrate unicast and multicast routing, a destination D is treated as a connected *destination mesh* containing one or more nodes. In the case of a unicast data flow, D is a singleton that contains a node with identifier D, whereas in the case of a multicast data flow, D contains the members of a multicast group, as well as a set of nodes needed to keep D connected. We refer to this set of nodes used to maintain D connected as the multicast mesh of D (MM_D).

STORM adopts the interest-driven approach introduced in [16]. However, unlike our prior work, the routing meshes used in STORM have the restriction that each node in those paths is *flow ordered*. As discussed in Section 6.1, a node is flow ordered for a given real-time flow if and only if the node can reserve a time slot (or set of time slots) at the appropriate position in the STORM frame as required by the end-to-end schedule of the flow. In the example of Fig. 3, a flow-ordered routing mesh consisting of a single path from S to D is shown. Elastic flows are routed using simple loop-free paths from sources to destinations, and they have no end-to-end restrictions, given that they do not have to be flow ordered. In the rest of this paper, unless otherwise stated, we refer to routing meshes for elastic or real-time flows simply as *routing meshes*.

Enclaves are used to confine the dissemination of signaling packets into connected regions of the network that contain those nodes with interest in a given data flow. The *enclave* of a destination D (E_D) is the union of the destination D, the set of active sources S, the routing meshes used to connect the elements of S with D, as well as nodes located one hop away from them. Once an enclave is established, it is used to restrict the dissemination of control information to those nodes that are part of the enclave.

Meshes and enclaves are activated and deactivated by the presence or absence of data packets and are initiated and maintained by the unicast destination or by a dynamically elected representative of the multicast destination.

The first source that becomes active for a given destination sends its first data packet piggybacked in a Mesh Request (MR) packet that is flooded up to a horizon threshold. If the interest expressed by the source spans more than the single data packet, the intended receivers of an MR start the process of establishing and maintaining its routing mesh and enclave. A mesh request (MR_D^S) generated by source S for destination D and transmitted by node B is a six-tuple of the form: ($horizon$, $persistent$, id^S, d_S^B, id^D, sn_S^B), where $horizon$ is an application-defined horizon threshold used to limit the dissemination of the MR, $persistent \in \{true, false\}$ is a flag that indicates the persistence of the interest, id^S is the sender's identifier, d_S^B is the distance from the sender, id^D is the destination's identifier (unicast destination or multicast group), and sn_S^B is an identifier for the message.

7.2 Information Stored and Exchanged

STORM uses mesh announcements to establish and maintain routing and destination meshes, to publish the availability of time slots in their corresponding flow ordered intervals, to coordinate end-to-end schedules for real-time flows and, in the case of a multicast group, to elect the core of the group. As long as there are active sources, the elected core or unicast destination periodically floods the enclave of the destination with MAs that contain monotonically increasing sequence numbers.

A *mesh announcement* (MA_D^{*B}) transmitted by node B for destination D is a eight-tuple of the form: (id^{*B}, $core_D^{*B}$, sn_D^{*B}, d_D^{*B}, mm_D^{*B}, $next_D^{*B}$, $slot_D^{*B}$, $ordered_D^{*B}$), where id^{*B} is the identifier of B, $core_D^{*B}$ is either the identifier of the core of the multicast group D known by B, or the identifier of the unicast destination, sn_D^{*B} is the largest sequence number known by B of destination D, d_D^{*B} is the distance of B to the core of D (the destination itself in the case of a unicast destination). The flag mm_D^{*B} is multicast-specific and indicates whether B is a mesh member (MM), a multicast group member, both, or a regular node (REG). The identifier $next_D^{*B}$ denotes the preferred next hop of node B toward the core of D. The time slot identifier $slot_D$ is randomly selected and is used in (1) as a reference to pick out the slots that will be reserved by the relays of a data flow. Lastly, $ordered_D^{*B}$ is a flag that indicates whether B has free slots in the interval defined by (1), i.e., if B is flow-ordered for destination D.

For a given destination D, a node maintains a neighborhood list L_D that stores an ordered set composed of the MAs that the node has recently received from its neighbors regarding that destination. In our notation, and to differentiate from an announcement that has just been received, an announcement received from neighbor B that is already stored in L_D is denoted as MA_D^B (with the $*$ dropped).

The announcements stored in L_D are also augmented with a time stamp (ts) obtained from the local clock and are ordered using a strict total order relation $\prec$ which is defined as follows:

$$MA_D^B \prec MA_D^A \Leftrightarrow (sn_D^B < sn_D^A) \vee (sn_D^B = sn_D^A \wedge d_D^B > d_D^A) \vee (sn_D^B = sn_D^A \wedge d_D^B = d_D^A \wedge id^B < id^A). \quad (2)$$

A node x also keeps track of the core of the destination ($core_D^x$) which is dynamic in the case of a multicast destination, the largest sequence number known for the destination (sn_D^x), its current distance to the core (d_D^x), its *feasible distance* to the core (fd_D^x), its preferred next hop toward the core ($next_D^x$), the reference slot identifier ($slot_D^x$), its mesh membership status flag mm_D^x that in the case of a unicast destination has no meaning, and a set of (source's address, δ)-pairs. Each pair indicates the direction in which x is forwarding the data packets of a given source-destination flow. In the case of a unicast destination δ always equals 1

and in the case of a multicast destination δ can take values in $\{1, -1\}$ depending on x's position in the routing meshes. The initial value of the routing state is as follows:

$$L_D \leftarrow \emptyset, core_D^x \leftarrow next_D^x \leftarrow nil, sn_D^x \leftarrow 0, d_D^x \leftarrow fd_D^x \leftarrow \infty, slot_D^x \leftarrow INVALID\_SLOT, mm_D^x \leftarrow REG$$

(regular node), and $\delta \leftarrow 1$.

7.3 Processing Mesh Announcements

Node x accepts a MA if it contains a sequence number equal to or larger than the current largest sequence number stored at node x, or if it is the first time that a MA is received from B (3). The MA is dropped otherwise

$$L_D \leftarrow \begin{cases} L_D \cup \{MA_D^{*B}\} & \text{if } MA_D^B \notin L_D \\ L_D - \{MA_D^B\} \cup \{MA_D^{*B}\} & \text{if } sn_D^x \leq sn_D^{*B} \\ L_D & \text{if } sn_D^x > sn_D^{*B}. \end{cases} \quad (3)$$

x's feasible distance to the core of D (fd_D^x) is a nonincreasing function over time that can only be reset by a change of core or by a new sequence number (4). Feasible distances are used to select a *feasible set* of next hops toward the core. The elements of a feasible set are not necessarily flow ordered

$$fd_D^x \leftarrow \begin{cases} d_D^{*B} & \text{if } sn_D^{*B} > sn_D^x \\ \min\{fd_D^A, d_D^{*B}\} & \text{if } sn_D^{*B} = sn_D^x \\ fd_D^x & otherwise. \end{cases} \quad (4)$$

The sequence number stored at node x for the core of destination D (sn_D^x) is a strictly increasing function over time that can only be reset by a change of core

$$sn_D^x \leftarrow \max\{sn_D^x, sn_D^{*B}\}. \quad (5)$$

The distance to the core of destination D of node x (d_D^x) is computed using (6) and the relation $\prec$ defined in (2). By definition, the core of the group has a 0 distance to itself and its feasible distance is always 0. In this paper, distances are measured in number of hops, and hence the cost (lc) of an existing link is 1. The core of a unicast destination is always the destination itself

$$d_D^x \leftarrow \begin{cases} d_D^i + lc_i^x : \max\limits_{i \in L_D : sn_D^i = sn_D^x} \{i\} & \text{if such } i \text{ exists} \\ \infty & otherwise. \end{cases} \quad (6)$$

The address of the next hop to the core of D ($next_D^x$) is also computed using the relation $\prec$ defined in (2), the current values of the feasible distance and sequence number, and the flow-ordered flag

$$next_D^x \leftarrow \begin{cases} id^i : \max\limits_{i \in CF_D^x} \{i\} & \text{if such } i \text{ exists} \\ nil & otherwise, \end{cases} \quad (7)$$

where

$$CF_D^x = \{i : i \in L_D \wedge fd_D^x = d_D^i \wedge sn_D^i = sn_D^x \wedge ordered_D^i = true\}$$

is the set flow-ordered feasible neighbors of node x for destination D. When routing elastic data packets, CF_D^x is relaxed to $F_D^x = \{i : i \in L_D \wedge fd_D^x = d_D^i \wedge sn_D^i = sn_D^x\}$, which is the set of x's feasible neighbors for D. By selecting next hops according to (7), STORM establishes loop-free directed meshes composed by nodes that have reported to have available time slots at the adequate positions in the STORM frame, and hence, that are capable of instantiating the flow's end-to-end schedule.

Lastly, if a node x receives a MA advertising a core with a larger identifier then L_D is set to $\{MA_D^{*B}\}$, $core_D^x$ is set to $core_D^{*B}$, and the other parameters are set as follows: fd_D^x to d_D^{*B}, d_D^x to $d_D^{*B} + lc_B^x$, sn_D^x to sn_D^{*B}, and $next_D^x$ to id^B and $slot_D^x$ to $slot_D^{*B}$. Otherwise, if $core_D^{*B} < core_D^x$, then the MA is simply discarded.

The mesh membership flag $mm_D^x \in \{RM, MM, RCV, REG, NIL\}$ indicates whether x is a regular node, a group receiver (RCV), a mesh member or both group receiver and mesh member (RM). As we have already mentioned, mm_D^x equals nil in the case of a unicast destination. A node x is a mesh member if and only if

$$\exists y \in L_D : mm_D^y \neq REG \wedge mm_D^y \neq NIL \wedge d_D^y > d_D^x \wedge next_D^y \leq id^x \wedge ts_D^y + MA\_period \geq ct, \quad (8)$$

where ts_D^y is the time stamp added to y when it was stored in L_D, ct is the current value of x's clock, and MA_period is the value of the MA-period.

The routing algorithm computes the end-to-end channel schedule of a data flow (it determines who and when data packets are forwarded), and the reservation protocol is only in charge of generating an instantiation of this schedule.

7.4 Transmission of Mesh Announcements

A node transmits MAs to inform other nodes about updates in its routing state. These updates can be originated by such internal events as a change in the group membership status (a node joining or leaving a multicast group) that modify the value of mm_D^x, or the generation of a new sequence number in the case of the core; or by an external event such as the reception of an MA_D^{*B} from neighbor B. Therefore, whenever the core of a destination generates a new MA with a larger sequence number, the latter is disseminated along the enclave advertising the new sequence number (5) and establishing next hop pointers toward the core (7). The mesh composed of these next hop pointers from a source to the core is called the *routing mesh* of that source.

In the case of a multicast destination, a MA transmitted by a multicast group member R, forces R's next hop (n) to update its mesh membership status according to (8). If this changes the value of mm_D^n, then n must transmit a new MA to advertise its new state. This way, nodes that lay in paths $p = R, n, n_1, \ldots, n_k, core_D$ with $next_D^R = n, next_D^n = n_1, \ldots, next_D^{n_k} = core_D$ are forced to become multicast mesh members, creating a connected component that contains all the receivers of a multicast group and that we have denominated as the *multicast destination*. The set of nodes $M^D = \{y : mm_D^y = MR \vee mm_D^y = MM\}$ form the *multicast mesh* of the multicast destination D.

7.5 Multicast Destinations and Core Elections

Upon reception of a MR, a multicast group member first determines whether it has received a MA from the core of that group within the last two MA-periods. If that is the case, no further action is needed; otherwise, the receiver considers itself the core of the group and starts transmitting

MAs to its neighbors, stating itself as the core of the group. Nodes propagate MAs based on the best MA they receive from their neighbors. An MA with a higher core id is considered better than one with a lower core id. Therefore, if a node receives a MA advertising a core with a larger id than the current core, then the new core is adopted and a new MA advertising the new core is transmitted. Eventually, each connected component has only one core.

7.6 Packet Forwarding

When a source has data to send, it checks whether it has received an MA advertising the intended destination within the last three MA periods. If that is the case, the sender simply broadcasts the data packet; otherwise, it broadcasts an MR.

Upon reception of a data packet, a node checks for a hit in its data-packet cache. If the *(sender's address, sequence number)* pair is already in the cache, the packet is silently dropped. Otherwise, the receiving node inserts the pair in its cache and determines whether it has to relay the data packet or not. If the node is part of the destination, it also passes the data packet to upper layers.

Equation (9) is used by node x to decide if it has to forward a data packet with destination D received from neighbor y.

$$mm_D^x = RM \vee mm_D^x = MM \vee \exists y \in L_D : \\ d_D^y > d_D^x \wedge next_D^y \leq id^x. \tag{9}$$

Equation (9) states that node x forwards a data packet received from node y if x is part of the multicast mesh or if x was selected by the previous relay (y) as one of its next hops to the core. This way, data packets travel along directed routing meshes until they reach either the first mesh member or the destination itself and then, in the case of a multicast destination, they are flooded along the multicast mesh. As we mentioned in Section 7.4, elastic data packets follow directed meshes composed of a single path and real-time data packets are routed using directed meshes composed of multiple flow-ordered paths. In the case of a unicast destination, the first two terms of (9) are always false; hence, nodes only forward data packets if they are part of a selected source-destination shortest path.

Lastly, if a node x receives a multicast data packet generated by a source s from a neighbor $y \in L_D$ it updates the value of δ in the (id^s, δ)-pair according to

$$\delta \leftarrow \begin{cases} 1 & \text{if } d_D^y > d_D^x \wedge next_D^y \leq id^x \\ -1 & \text{if } d_D^y \leq d_D^x \wedge mm_D^y = RM \vee mm_D^y = MM \wedge \\ & mm_D^x = RM \vee mm_D^x = MM. \end{cases} \tag{10}$$

This way, nodes adapt their end-to-end schedules according to their positions with respect to the core of the multicast group and the multicast sources.

8 Correctness in STORM

We show that no directed loops can be formed in the nodal routing tables, and that the end-to-end reservations established along routing meshes attain bounded end-to-end delays for real-time traffic. In our proofs, we assume that the network is connected, and that distances are measured in number of hops and hence a link cost (lc) is either 1 or infinity. Let the current routing state stored at node n_i regarding destination D be $s_D^{n_i} = (id^{n_i}, core_D^{n_i}, sn_D^{n_i}, d_D^{n_i}, fd_D^{n_i}, mm_D^{n_i}, next_D^{n_i})$, and let $\prec_s$ be a total order relation defined in the same way as the relation of (2) but over the routing state of nodes. In the case of a unicast destination, $core_D^{n_i}$ stores the address of the destination itself.

Theorem 1. *Any successor path $p = \{n_0, n_1, \ldots, n_k\}$ with $next_D^{n_0} = n_1$, $next_D^{n_1} = n_2, \ldots, next_D^{n_{k-1}} = n_k$ established using the data structures and procedures described in Section 7.2 is loop-free.*

Proof. From (2) to (7) we have that $next_D^{n_i} = n_j \Longrightarrow s_D^{n_i} \prec_s s_D^{n_j}$ and hence, for any path $p = \{n_0, n_1, \ldots, n_k\}$ we also have $s_D^{n_0} \prec_s s_D^{n_1} \prec_s \cdots \prec_s s_D^{n_{k-1}} \prec_s s_D^{n_k}$. Now, let us proceed by contradiction and assume that a loop is formed in p when a node n_i selects n_x as its next hop. Then, we would have $s_D^{n_x} \prec_s s_D^{n_{i-j}} \prec_s \cdots \prec_s s_D^{n_i} \prec_s s_D^{n_x}$, which is a contradiction. □

The proof of Theorem 1 makes the implicit assumption that the distances from nodes to the destination cannot be increased for a given sequence number. Theorem 2 relaxes this assumption by allowing nodes to remove neighbors from their L_D and then update d_D^x according to (6).

Theorem 2. *Considering the case of Theorem 1, let $p = \{n_0, n_1, \ldots, n_k\}$ be a successor path with $next_D^{n_0} = n_1$, $next_D^{n_1} = n_2, \ldots, next_D^{n_{k-1}} = n_k$, then any change of successor along that path does not create loops.*

Proof. The proof is by contradiction. Assume that a cycle is formed when node n_i selects n_x (after removing n_y from its neighborhood list L_D or after receiving an update that disqualifies n_y as a feasible successor) as its next hop. It follows from (7) that, for n_x to be selected by n_i as its next hop, n_x has to be an element of the feasible set of next hops of n_i for destination D, i.e., an element of $F_D^{n_i} = \{n : fd_D^{n_i} = d_D^n \wedge sn_D^n = sn_D^{n_i}\}$. Now, if $n_x \in F_D^{n_i}$ and given that $lc_{n_x}^{n_i} > 0$, we have $d_D^{n_x} = fd_D^{n_i} < d_D^{n_i}$. Therefore, either $s_D^{n_i} \prec_s s_D^{n_x}$ or $next_D^{n_i} = n_x = nil$. In the first case, the same contradiction of Theorem 1 is reached, i.e., that $s_D^{n_x} \prec_s \cdots \prec_s s_D^{n_i} \prec_s s_D^{n_x}$. In the second case, it follows that a cycle with a nil element must exist, which is also a contradiction. Therefore, the theorem is true. □

Now, let $p = \{n_1, n_2, \ldots, n_D\}$ be a flow-ordered successor path of length l with $next_D^{n_1} = n_2, next_D^{n_2} = n_3, \ldots, next_D^{n_{l-1}} = n_D$ as computed by STORM following (7), and let n_D be the destination of a packet transmitted by n_1. If all nodes in p reserve a slot in the interval $[(slot_D - \delta d_D^{n_i}\Delta) \bmod N, ((slot_D - \delta d_D^{n_i}\Delta) \bmod N + \Delta) \bmod N)$ where $slot_D$ is any slot identifier, $d_D^{n_i}$ is the distance in hops from a node n_i to n_D as computed by (6) and $\Delta < N$, then the following theorems can also be proved:

Theorem 3. *The maximum end-to-end delay experienced by a real-time data packet that fits in a time slot and is transmitted either from n_1 to n_D or from n_D to n_1 is $\Delta\tau l$.*

Proof. It follows from (4), (6) and (7) that a node n_i can select n_{i+1} as its next hop to n_D if and only if $d_D^{n_n} + 1 = d_D^{n_{n+1}}$. Hence, if $next_D^{n_i} = n_{i+1}$, then n_i must select a slot in the

range $[(slot_D - (\delta d_D^{n_{i+1}} + 1)\Delta) \bmod N,\ ((slot_D - (\delta d_D^{n_{i+1}} + 1)\Delta) \bmod N + \Delta) \bmod N)$ when establishing a reservation, while n_{i+1} must select a slot in the range

$$[(slot_D - \delta d_D^{n_{i+1}}\Delta) \bmod N, ((slot_D - \delta d_D^{n_{i+1}}\Delta) \bmod N + \Delta) \bmod N).$$

If the data packet is transmitted from n_1 to n_D the value of δ for both nodes n_i and n_{i+1} equals 1 (10) and if the data packet is transmitted from n_D to n_1 the value of δ for both nodes n_i and n_{i+1} equals -1 (10). In both cases, the maximum distance between any two slots selected from these intervals is 2Δ. Following this reasoning, the end-to-end delay is increased by a maximum of Δ slots by adding an extra next hop to the path. From Theorem 2 we know that any successor path p established by STORM is loop-free; hence, a packet that follows p will take at most $\Delta\tau l$ seconds to reach its destination.

We have to show that another real-time flow, say f_x, that intersects flow f_{1D} at one or more nodes cannot affect the end-to-end delay of a packet of f_{1D}. We proceed by contradiction. Assume that, at node $n_i \in p$, a packet of flow f_x is transmitted instead of a packet of f_{1D} during a slot that was reserved on behalf of flow f_{1D} (increasing the delay experienced by the packet of flow f_{1D}). From Section 4 we know that the priority of f_{1D}'s queue at this particular slot is higher than the priority of any other data queue. Therefore, no data packet could have been extracted from a queue different to f_{1D}'s queue. □

Theorem 3 holds even in the case of a multicast flow following a path p that traverses its multicast mesh to reach the core of the group. This is true because all the nodes in p use (10) to set their values of δ to 1. Lastly, Theorem 4 characterizes the delay experienced by a real-time multicast data packet.

Theorem 4. *The maximum end-to-end delay experienced by a multicast real-time data packet that fits in a time slot and is transmitted from n_1 to the multicast group D is $\Delta\tau(l + m)$ seconds, where l is the length of a successor path connecting the source n_1 with the core c of D and, m is the length of the longest successor path consisting of mesh members connecting any multicast receiver to c.*

Proof. We have two cases. In the first case, the successor paths that connect n_1 to the destination reach the core without touching any other mesh member. From Theorem 3 we know that the longest it can take a data packet to reach c is $\Delta\tau l$ seconds. Let r be the receiver connected to c by the longest successor path $p_l = \{r, n_2, \ldots, c\}$. From (9) we know that nodes in p_l must forward the packet transmitted by n_1, and from (10) we know that the value of δ for source n_1 in all these nodes equals -1. Hence, the longest it can take a data packet to travel from c to r is $\Delta\tau m$ seconds (Theorem 3). Therefore, the maximum end-to-end experienced by a data packet is $\Delta\tau l + \Delta\tau m$. In the second case, the successor paths that connect n_1 to the destination reach a set of mesh members before reaching the core c. From (10) we know that the value of δ equals 1 in all the nodes that lay in successor paths from n_1 to c; therefore, the delay experienced by the data packets to reach the core c is at most $\Delta\tau l$. Lets consider an arbitrary successor path $p = \{r, m_1, \ldots, c\}$ composed of mesh members that connect a receiver r to the core c and let m_i be the first node in p that receives the data packet from a node m_j closer to the core (i.e., a data packet flowing away from the core and toward a receiver). If no such node exists, then r received the packet when the latter was traveling toward the core and hence, its delay is less than $\Delta\tau l$ (Theorem 3). Given that m_j is either the core or is relaying the packet toward the core, it follows from Theorem 3 that it relays the data packet at most $\Delta\tau(l + 1)$ seconds after it was originated by n_1. Moreover, because δ equals -1 in m_i (10), the interval computed by node m_i using (1) starts at least $\Delta\tau d_D^{m_i}$ seconds after the beginning of the interval of the core. Therefore, m_i already has the packet generated by the source at the beginning of its own flow-ordered interval, which starts $\Delta\tau(l + 1 + d_D^{m_i})$ seconds after the data packet was generated by the source. Lastly, we can use Theorem 3 to argue that the packet will take at most $\Delta\tau(m - d_D^{m_i} - 1)$ more seconds to reach r. □

9 Performance Results

We present simulation results comparing STORM with ODMRP for the case of multicast traffic, as well as AODV and OLSR for the case of unicast traffic. In our experiments, ODMRP, AODV, and OLSR run on top of IEEE 802.11 DCF and all the protocols use a 802.11b physical layer. We selected these protocols because they have become *de facto* baselines for performance comparisons of multicast, unicast, and channel access protocols. Even though they were not designed for real-time traffic, they are a good reference that allows us to highlight the performance gains of our approach. We use packet delivery ratio, generalized group delivery ratio, end-to-end delay, and total overhead as our performance metrics. To measure total overhead, we count *all* the packets generated by each protocol stack, which for the case of STORM includes data packets, MRs, MAs, hellos, and reservation packets. The generalized group delivery ratio is a multicast-specific metric in which a data packet is considered as delivered, if and only if it is received by at least a given proportion of the multicast group members. This metric emphasizes the importance of group delivery by not considering packets that are received by a small subset of the group members. For this paper we set a threshold of 80 percent. The total overhead is computed as the average total number of packets transmitted by each node.

We employ random waypoint (RW) and a combination of random waypoint and group mobility [8] models as our mobility models. In our combined scenarios, the members of a given multicast group move following the group mobility model, whereas nodes that do not belong to a multicast group move according to the RW mobility model. This combined mobility model depicts more accurately common situations where the members of the same rescue team or military patrol tend to move close by. The node speeds used in the simulation experiments vary from 1 to 20 m/s (or 3.6 to 72 Km/h) to cover pedestrian and vehicular speeds. This range of speeds is the same that has been used in the past by

TABLE 1
Simulation Environment

Total nodes	100	Node placement	Random
Simulation area	$1800 \times 1800 \text{m}^2$	Simulation time	150s
Phy. Model	802.11b	Tx. power	15dbm
		Tx. Rate	11000000bps
Data source	MCBR and CBR	Pkts. per src.	1000
Mobility model	Random waypoint (RW)	Pause time	10s
		Min.-Max. Vel.	1-20m/s
Mobility model	Group mobility	Node pause time	10s
		Grp. pause time	10s
Grp. Min-Max Vel.	1-20m/s	Node Min-Max Vel.	1-20m/s

prior work for the analysis of routing and dynamic transmission scheduling in dynamic ad hoc networks.

We used the discrete event simulator Qualnet [19] version 3.9, that provides a realistic simulation of the physical layer, and well tuned versions of ODMRP, AODV, OLSR, and IEEE 802.11 DCF, which we call "WiFi" for simplicity. Each simulation was run for 20 different seed values. All the multicast protocols use the same period of 3 seconds to refresh their routing structures (join query period for ODMRP and announcement periods for STORM). STORM's frame size is set to 200 slots and each slot has a duration of 0.5 ms. The value of Δ introduced in Section 6.1 is set to 20, and was manually set following the rationale described in Section 6.1. For ODMRP, the forwarding group time-out was set to three times the value of the join query period, as advised by its designers. Table 1 lists the details of the simulation environment.

Because of space limitations, we do not show results for networks subject only to unicast traffic, and discuss only simulation results for multicast and combined traffic. However, the results observed in simulations for unicast traffic follow the same pattern we discuss for the case of networks subject to unicast and multicast traffic.

9.1 Multicast Traffic

In these experiments, each MCBR source transmits 10 packets of 200 bytes (which is the output size of the G.711 VoIP coder [10]) per second and the multicast group is composed of 20 nodes. In the first set of experiments nodes move around inside of a mobile square region of $900 \times 900 \text{ m}^2$ and the remaining 80 nodes move following the random waypoint mobility model. In the second set of experiments, the totality of the nodes move following the random waypoint mobility model. Sources are not group members. For STORM simulations, one out of three sources is defined as real time. For instance, in a scenario with six MCBR sources, two of them are defined as real time and the other four are defined as elastic. The selection of the real-time sources is random.

Figs. 4a, 4b, 4c, and 4d present results for a scenario where the number of MCBR sources is increased from 6 to 24. The curves labeled as "STORM RT" present the average results taken over the real-time flows only, the graphs labeled as "STORM elastic" present the average results taken over the elastic flows only, and the graphs labeled "STORM elastic+RT" present the average results taken over the totality of the flows. The curves labeled with "RWP" show results obtained for a scenario in which all the nodes move according to the RW mobility model.

It is clear from Fig. 4a that STORM RT outperforms ODMRP+WiFi for all the values of the number of MCBR sources in terms of delivery ratio. In particular, STORM RT delivers 25 percent more packets than ODMRP for 18 sources

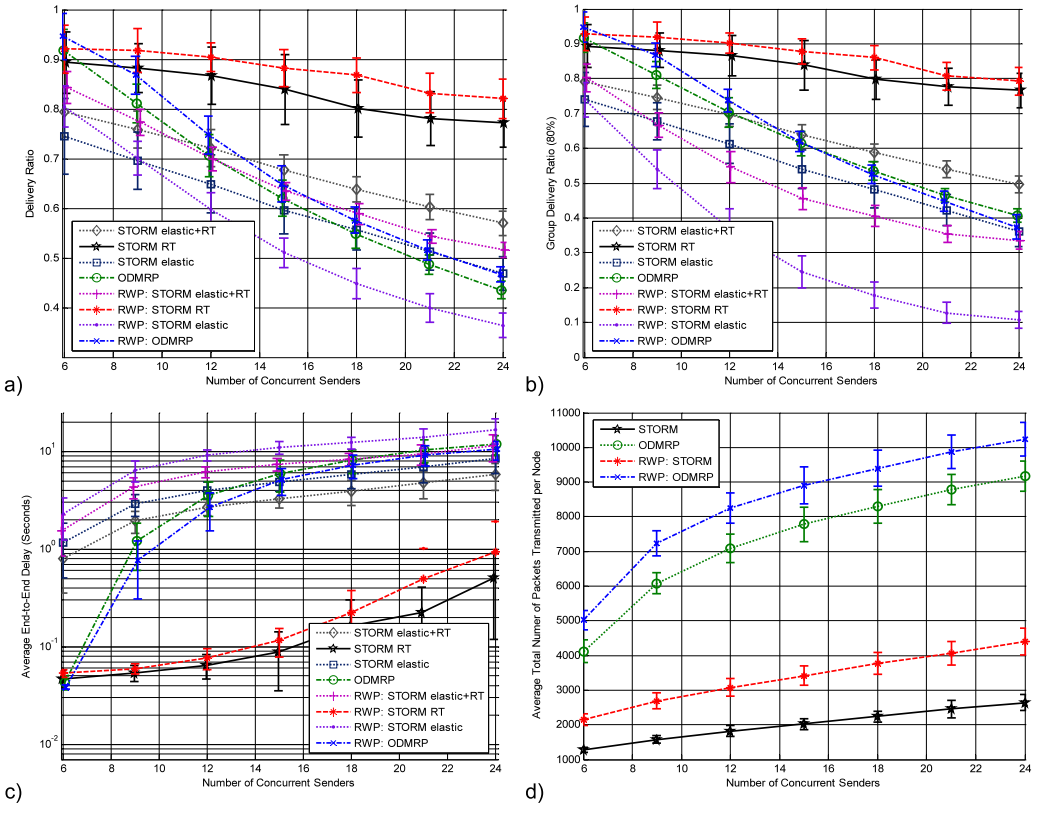

Fig. 4. Performance with increasing number of MCBR sources. (a) Delivery ratio. (b) Group delivery ratio. (c) End-to-End delay. (d) Total overhead.

or more. On the other hand, for up to nine sources ODMRP delivers more packets than STORM elastic+RT. However, as the number of sources increases, the delivery ratio of ODMRP drops faster and ends up delivering 15 percent fewer packets than STORM elastic+RT and 5 percent fewer packets than STORM elastic. Fig. 4b depicts a similar behavior for the case of group delivery ratio. For 24 sources, STORM RT delivers 30 percent more packets to at least the 80 percent of the receivers than ODMRP and for more than 15 sources STORM elastic+RT and STORM elastic perform similar or better than ODMRP.

Fig. 4c shows the end-to-end delay attained by the protocols using a logarithmic scale. STORM RT attains delays that are an order of magnitude lower than in ODMRP+WiFi for the case of nine or more sources. Moreover, the delays attained by STORM RT comply with the International Telecommunication Union (ITU) Recommendation G.114 [11], which considers end-to-end delays between 0-150 msec as acceptable for most user voice applications. In contrast, the traditional MANET protocol stack incurs delays in the order of seconds, which are unacceptable for voice.

Fig. 4d presents the total overhead induced by the protocols. From the figure, we observe that STORM is much more efficient than the traditional protocol stack with ODMRP. It consistently generates less than one third of the packets generated by the traditional protocol stack with ODMRP.

Figs. 4a, 4b, 4c, and 4d show that "RWP: STORM RT" attains very good performance even in the scenario where the multicast receivers can be spread across the whole simulation area, nodes can move independently of one another, and the routing structures that connect sources to receivers tend to be much larger and dynamic than those that appear in the case of the group mobility model.

The superior performance of STORM compared to the traditional MANET protocol stack based on WiFi in these experiments can be explained by its use of an efficient distributed transmission scheduling scheme and resilient meshes over which traffic flows free of collisions from sources to destinations even when the network topology changes. The data-flow-driven reservation scheme in STORM allocates more bandwidth to those nodes that in a given point in time are relaying more data packets for one or more data flows. This bandwidth allocation strategy helps reduce the number of packets dropped in the node queues and the queuing waiting time, which is critical for end-to-end delays. In contrast, with the traditional MANET protocol stack, the probability of control packets colliding with other packets increases with traffic load and topology changes, repeated collisions force real time and elastic traffic to back off, and relay nodes with long queues compete on an equal basis with nodes having much less traffic to relay. In addition, collisions result in the loss of signaling packets, which is interpreted by the routing protocols as the loss of routes and triggers more signaling overhead. All of this is detrimental to real-time flows, and the performance of the routing protocols operating independently of a contention-based channel access scheme. As traffic increases with the number of sources, more links are perceived as failing and the routing protocols need to find new routes, which induces even more traffic load due to the extra signaling overhead generated by the routing protocols.

9.2 Combined Traffic

This scenario focuses on combined multicast and unicast traffic. The number of multicast groups is increased from one to six with three concurrent active sources per multicast group. As in the previous experiment, one of these sources is defined as real time and the remaining two are elastic, and the selection of the real-time source is random. In these experiments, sources are also group members, which favors ODMRP. In addition to the multicast flows, we have five concurrent CBR flows among nodes that are randomly selected from nodes that are not part of any multicast group. Multicast sources and CBR sources send a total of 1,000 data packets of 200 bytes at a rate of 10 packets per second. For STORM, the five unicast flows are defined as real time. As in the previous experiment, group members follow the group mobility model with group regions of $900 \times 900\ \mathrm{m}^2$, whereas the remaining nodes move according to the random waypoint model.

Fig. 5a shows the packet delivery ratio attained by STORM and by OLSR and AODV when they are running in parallel with ODMRP. The results show that STORM multicast outperforms ODMRP even for the case of the elastic flows, and that STORM RT scales quite well as the number of multicast groups increases. Regarding unicast routing, we observe that AODV performs similar to STORM when the network has up to four multicast groups. However, as the number of multicast groups increases and the network is more heavily loaded, STORM unicast clearly outperforms both OLSR and AODV by delivering more than twice as many packets as OLSR does, and up to 30 percent more packets than AODV does. Fig. 5a shows that running unicast protocols in parallel with an independent multicast routing protocol induces considerable overhead, and that the integrated signaling in STORM is very effective.

As Fig. 5b shows, STORM performs better than ODMRP in terms of the group delivery ratio even for elastic traffic, and that the group delivery ratio attained by STORM is up to 20 percent higher than that of ODMRP for real-time traffic. Fig. 5c shows the end-to-end delay attained by the different protocols. As the number of sources increases, the average delay of STORM unicast is an order of magnitude smaller than that of AODV, which in turn is an order of magnitude smaller than the one attained by OLSR. The delays attained by STORM unicast comply with the G.114 recommendation of the ITU-T. For the case of multicast traffic, the end-to-end delay attained by STORM RT is also an order of magnitude smaller than that of ODMRP and also comply with the G.114 recommendation. Lastly, Fig. 5d presents the total overhead induced by the different protocols. With STORM, nodes transmit as few as half of the packets transmitted by the nodes running a traditional protocol stack.

10 Conclusions

We introduced STORM, a cross-layer protocol framework for wireless ad hoc networks that integrates interest-driven

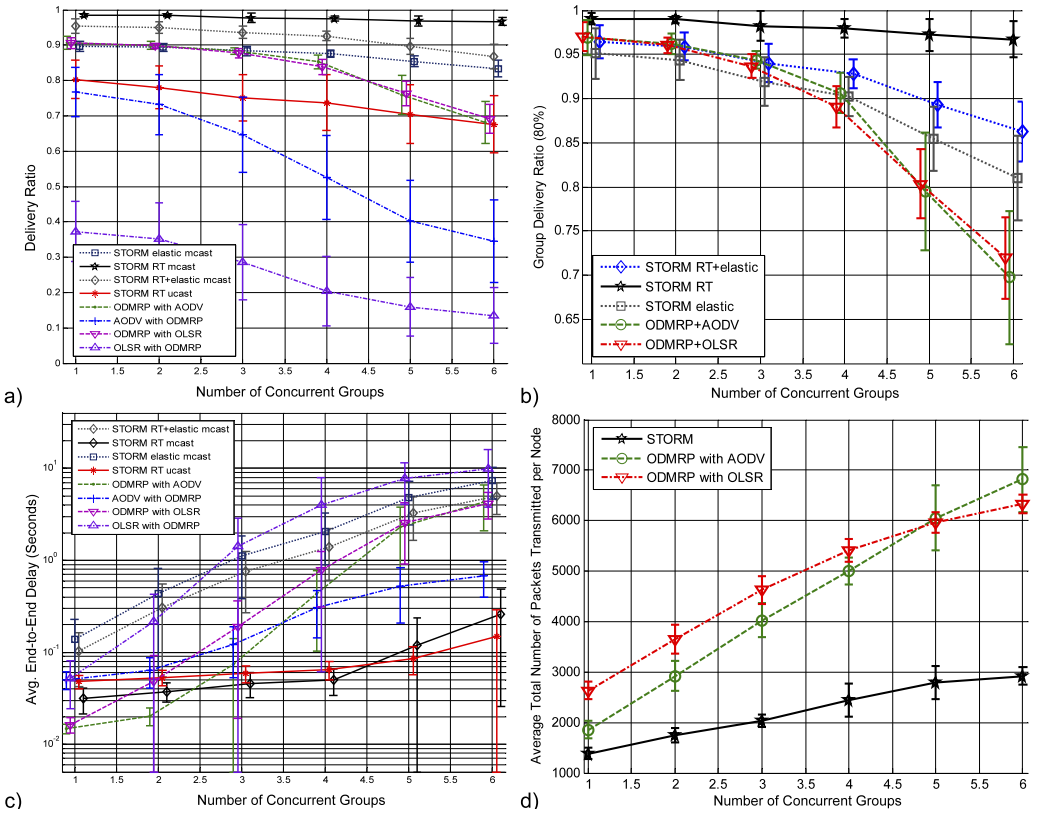

Fig. 5. Performance with increasing number of active groups, three MCBR sources per group, and five unicast flows. (a) Delivery ratio. (b) Group delivery ratio. (c) End-to-End delay. (d) Total overhead.

routing with priority-based queuing for traffic management, end-to-end bandwidth reservations controlled by the routing, and distributed transmission scheduling. All these components work together to provide end-to-end delay and bandwidth guarantees to real-time unicast and multicast data flows in multihop wireless networks even when nodes move. We proved that the routing meshes established with STORM are loop-free at any time and that the end-to-end reservations established along routing meshes provide bounded delays to real-time data packets. Our simulation results confirm our correctness results showing that STORM is very scalable and robust for both unicast and multicast traffic. The results also show that STORM scales better than the traditional protocol stack, which consists of the IEEE 802.11 DCF working independently of the routing protocols (AODV, OLSR, and ODMRP), and that the end-to-end delays attained with STORM comply with the ITU-T recommendation G.114 that describes the delay characteristics needed to support voice applications. Arguably, STORM's main limitation is the need for time-slotted channel access requiring clock synchronization; however, viable approaches exist to attain this.

Acknowledgments

This work was sponsored in part by the US Army Research Office (ARO) under grant W911NF-05-1-0246, by the Baskin Chair of Computer Engineering, by the UC MEXUS-CONACyT program, and by the Mexican National Polytechnic Institute (IPN).

References

[1] L. Bao and J.J. Garcia-Luna-Aceves, "A New Approach to Channel Access Scheduling for Ad Hoc Networks," *Proc. ACM MobiCom*, pp. 210-221, 2001.

[2] L. Bao and J.J. Garcia-Luna-Aceves., "Receiver-Oriented Multiple Access in Ad Hoc Networks with Directional Antennas," *Wireless Networks*, vol. 11, nos. 1/2, pp. 67-79, 2005.

[3] Z. Cai, M. Lu, and X. Wang, "An End-to-End Bandwidth Allocation Algorithm for Ad Hoc Networks," *Telecomm. Systems*, vol. 22, no. 1, pp. 281-297, 2003.

[4] E. Carlson, C. Prehofer, C. Bettstetter, H. Karl, and A. Wolisz, "A Distributed End-to-End Reservation Protocol for IEEE 802.11-Based Wireless Mesh Networks," *IEEE J. Selected Areas in Comm.*, vol. 24, no. 11, pp. 2018-2027, Nov. 2006.

[5] L. Chen and W.B. Heinzelman, "A Survey of Routing Protocols that Support QoS in Mobile Ad Hoc Networks," *IEEE Network*, vol. 21, no. 6, pp. 30-38, Nov./Dec. 2007.

[6] P. Djukic and P. Mohapatra, "Soft-TDMAC: Software TDMA-Based MAC over Commodity 802.11 Hardware," *Proc. IEEE INFOCOM*, 2009.

[7] W. Feller, *An Introduction to Probability Theory and Its Applications*, vol. 1, second ed. John Wiley, 1957.

[8] X. Hong, M. Gerla, G. Pei, and C.-C. Chiang, "A Group Mobility Model for Ad Hoc Wireless Networks," *Proc. ACM/IEEE Second Int'l Workshop Modeling, Analysis and Simulation of Wireless and Mobile Systems (MSWiM '99)*, pp. 53-60, 1999.

[9] *IEEE Std 802.11: Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) Specifications*, IEEE, pp. i-445, Nov. 1997.

[10] "ITU-T Recommendation G.108 (09/99) Application of the E-Model: A Planning Guide," http://www.itu.int/rec/t-rec-g.108-199909-i/en, 2012.

[11] "ITU-T Recommendation G.114 (05/03) One-Way Transmission Time," http://www.itu.int/rec/t-rec-g.114-200305-i/en, 2012.

[12] P. Jacquet, A. Laouiti, P. Minet, and L. Viennot, "Performance Analysis of OLSR Multipoint Relay Flooding in Two Ad Hoc Wireless Network Models," *Proc. Second IFIP-TC6 Networking Conf.*, May 2002.

[13] S.-J. Lee, M. Gerla, and C.-C. Chiang, "On-Demand Multicast Routing Protocol," *Proc. IEEE Wireless Comm. and Networking Conf. (WCNC)*, vol. 3, pp. 1298-1302, 1999.
[14] C.R. Lin and M. Gerla, "Asynchronous Multimedia Multihop Wireless Networks," *Proc. IEEE INFOCOM*, vol. 1, pp. 118-125, Apr. 1997.
[15] T. Melodia, M.C. Vuran, and D. Pompili, "The State of the Art in Cross-Layer Design for Wireless Sensor Networks," *Proc. EURO-NGI Workshops Wireless and Mobility*, pp. 78-92, 2006.
[16] R. Menchaca-Mendez and J.J. Garcia-Luna-Aceves, "An Interest-Driven Approach to Integrated Unicast and Multicast Routing in MANETs," *Proc. IEEE 16th IEEE Int'l Conf. Network Protocols (ICNP '08)*, pp. 248-257, Oct. 2008.
[17] C.E. Perkins and P. Bhagwat, "Highly Dynamic Destination-Sequenced Distance-Vector Routing (DSDV) for Mobile Computers," *Proc. ACM SIGCOMM*, pp. 234-244, 1994.
[18] C.E. Perkins and E.M. Royer, "Ad-Hoc On-Demand Distance Vector Routing," *Proc. Second IEEE Workshop Mobile Computing Systems and Apps. (WMCSA '99)*, pp. 90-100, Feb. 1999.
[19] Qualnet 3.9, Scalable Network Technologies, http://www.scalablenetworks.com, 2012.
[20] E. Setton, T. Yoo, X. Zhu, A. Goldsmith, and B. Girod, "Crosslayer Design of Ad Hoc Networks for Real-Time Video Streaming," *IEEE Wireless Comm.*, vol. 12, no. 4, pp. 59-65, Aug. 2005.
[21] Y. Wu, P.A. Chou, Q. Zhang, K. Jain, W. Zhu, and S.-Y. Kung, "Network Planning in Wireless Ad Hoc Networks: A Cross-Layer Approach," *IEEE J. Selected Areas in Comm.*, vol. 23, no. 1, pp. 136-150, Jan. 2005.
[22] J. Yuan, Z. Li, W. Yu, and B. Li, "A Cross-Layer Optimization Framework for Multihop Multicast in Wireless Mesh Networks," *IEEE J. Selected Areas in Comm.*, vol. 24, no. 11, pp. 2092-2103, Nov. 2006.

J.J. Garcia-Luna-Aceves received the BS degree in electrical engineering from the Universidad Iberoamericana, Mexico City, in 1977, and the MS and PhD degrees in electrical engineering from the University of Hawaii at Manoa, Honolulu, in 1980 and 1983, respectively. He holds the Jack Baskin Endowed chair of Computer Engineering at the University of California, Santa Cruz (UCSC), is chair of the Computer Engineering Department, and is a principal scientist at the Palo Alto Research Center (PARC). Prior to joining UCSC in 1993, he was a center director at SRI International in Menlo Park, California. He has been a visiting professor at Sun Laboratories in Menlo Park, California, and a principal of Protocol Design at Nokia in Mountain View, California. He received the IEEE Computer Society Technical Achievement Award in 2011 and is listed in Marquis *Who's Who in America* and *Who's Who in The World*. He was the corecipient of the IEEE Fred W. Ellersick 2008 MILCOM Award for best unclassified paper. He was also the corecipient of the Best Paper Awards from the European Wireless Conference 2010, IEEE MASS 2008, SPECTS 2007, IFIP Networking 2007, and IEEE MASS 2005 conferences, and of the Best Student Paper Award of the 1998 IEEE International Conference on Systems, Man, and Cybernetics. He received the SRI International Exceptional-Achievement Award in 1985 and 1989. He holds 36 US patents and has published more than 420 papers. He is a fellow of the IEEE, ACM, and AAAS.

Rolando Menchaca-Mendez received the BS degree in electronic engineering from the Universidad Autonoma Metropolitana, Mexico City, in 1997, the MS degree from the Mexican National Polytechnic Institute, Mexico City, in 1999, and the PhD degree in computer engineering from the University of California at Santa Cruz in 2009. He is a professor and head of the Communications and Networking Laboratory at the Computer Research Center (CIC) of the Mexican National Polytechnic Institute (IPN).