ROME: Routing Over Mobile Elements in WSNs

Stefano Basagni
ECE Department
Northeastern University
basagni@ece.neu.edu

Michele Nati, Chiara Petrioli and Roberto Petroccia
Dipartimento di Informatica
Università di Roma "La Sapienza"
{nati,petrioli,petroccia}@di.uniroma1.it

***Abstract*—In this paper we present ROME, a geographic routing protocol for wireless sensor networks (WSNs) with mobile nodes. ROME design is suited to deal with communication problems in WSN scenarios with high network dynamics, such as nodal addition, nodal removal and node mobility. In addition, it retains desirable properties of protocols for static WSNs such as using cross-layer techniques for performance optimization, dealing with asynchronous nodal duty cycles, and being able to deal with connectivity dead ends. We define the protocol in details and provide detailed simulation-based performance evaluation of ROME. In scenarios with static and mobile nodes together, our ns2-based experiments show that ROME performs remarkably well with respect to metrics such as packet delivery ratio, energy consumption and end-to-end packet latency.**

I. Introduction

Research in *wireless sensor networks* (WSNs) has been focusing on designing protocols at the different levels of the communications protocol stack that can provide energy efficient and cost effective solutions for the delivery of sensed data from a large number of *static* sensors to one or few *static sinks* (data collection points). A host of MAC and routing algorithms have been proposed that solve the problem of data dissemination in WSNs from different perspectives and for a wide variety of scenarios [1], [2]. Experiments on real-hardware testbeds, however, have shown that even if the nodes are static, WSNs exhibit high dynamics because of the instability of communication links and the disappearance of some of the nodes due the depletion of their energy and to their failure. New nodes can also be added to the network at any time, which also causes variable network connectivity. More important, recent applications for WSNs are gaining attention where some of the sensor nodes are mobile, being worn by humans, carried by animals, or mounted on mobile robots and vehicles. The vast majority of the solutions proposed for WSNs cannot effectively deal with this kind of network dynamics, and to this day, only a handful of solution have been presented for *mobile WSNs* (see [3]–[5] also for further references).

In this paper we present ROME, for *Routing Over Mobile Elements*, a geographic protocol for WSNs characterized by high dynamics, such as node addition, node removal and the mobility of some of the network nodes. ROME builds on geographic forwarding protocols (such as ALBA–R [6]) that have been defined for networks with static nodes and a static sink. From these protocols ROME retains desirable properties such as optimized performance through cross layer design, receiver-based relay selection, asynchronous nodal duty cycles, and the ability to deal with connectivity holes in the network topology (dead ends). To the best of our knowledge, no complete solution has been presented in the literature that, besides energy conservation, takes into account load balancing, nodal duty cycle and mobility as ROME does. In other words, while there are plenty of works for WSN routing in static scenarios and for deployments with mobile sinks, we were not able to find, in the current literature, protocols concerning the scenario considered here. The aim of this paper is that of defining ROME and demonstrating its effectiveness through extensive simulations especially geared to show that it is able to effectively support mobility and network dynamics. Through the designed experiments we were able to observe that mobile nodes stopping at or leaving from a given location do not significantly degrade performance in terms of packet delivery ratio, end-to-end latency and energy consumption. Our investigation has considered different percentages of mobile nodes, slow and fast mobility and different traffic conditions (from light to heavy). Comparison with solutions previously introduced in the literature for ad hoc networks (such as GPSR [7], flooding and probabilistic flooding protocols) has shown that ROME is more lightweigh, and achieves one order of magnitude increased energy efficiency at the cost of a limited increase in latency. The protocol is also more scalable, being able to deliver all generated packets at medium to high traffic whereas the other protocols get congested.

The rest of the paper is organized as follows. Section II provides an operational description of ROME. In Section III we provide an assessment of ROME performance through simulations that include comparisons with flooding and GPSR. Finally, Section IV concludes the paper.

II. ROME: Protocol Description

ROME combines an energy-efficient MAC protocol with greedy forwarding endowed with a mechanism for escaping from connectivity holes. The functioning of the protocol is summed up as follows. In order to save energy, nodes follow asynchronous awake/asleep schedules (duty cycles). A node does not know its neighbors and their duty cycles. When a node has a packet to transmit it initiates a contention phase, looking for a relay between its awake neighbors (eligible relays). The aim is that of finding a relay to advance the packet toward the sink. Let x be a node engaged in packet forwarding, and let $F(x)$ be the portion of node x transmission

978-1-4244-4148-8/09/$25.00 ©2009
This full text paper was peer reviewed at the direction of IEEE Communications Society subject matter experts for publication in the IEEE "GLOBECOM" 2009 proceedings.

area where relays offering positive advancement toward the sink are located (Figure 1). To advance the packet toward the sink node x selects a relay among the awake neighbors in $F(x)$ (greedy forwarding). This advancement is always

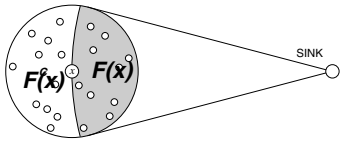

Figure 1. ROME: Forwarding regions of node x

possible when there is at least a neighbor closer to the sink. There are times, however, when this simple greedy mechanism fails because of *dead ends*, i.e., nodes that have no neighbors in the direction of the sink. To cope with dead ends ROME borrows the basic ideas of ALBA-R coloring mechanism, extending it to cope with network dynamics and nodes mobility (schemes à la ALBA-R work only for static networks). In ALBA-R the forwarding choices depend on the color that the nodes dynamically assume according to their perceived ability to forward packets directly to the sink. Let $C_0, \ldots, C_{max}$ indicate the *colors* a node can assume. Nodes with an even color search for neighbors in F (toward the sink), whereas nodes with odd color search for neighbors in F^C (away from the sink: See Figure 1). Nodes with color C_k can volunteer as relays only for nodes with color C_k or C_{k+1}, and can only look for relays with color C_{k-1} or C_k. This means that a node with color C_0 functions according to the basic advancement rules of greedy forwarding. It can reach the sink with a path made up only of nodes closer to the destination than itself. In other words, once a C_0 node is reached, the path to the sink is made up only of C_0 nodes. Similarly, packets generated or relayed by C_k nodes follow routes which first traverse C_k nodes, then go through C_{k-1} nodes, then through C_{k-2} nodes, and so on, finally reaching C_0 nodes, through which they greedily reach the sink. Each time the selected relay has a color different from the sender the forwarding changes direction. The distributed algorithm run by ALBA-R nodes to select the (minimum) color that allows them to successfully deliver packets to the sink is detailed in [6]. The protocol has been formally proven capable of always finding routes to the sink (if any) and it has been shown resilient to localization errors [8]. However, as mentioned, the current coloring rules makes it suitable for WSNs whose nodes do not move. ROME borrows from ALBA-R the idea of associating different forwarding choices to nodes depending on their color. However, it significantly changes the way the actual coloring is performed. As a result it successfully defines ways to tackle the problems introduced by nodal dynamics, such as node addition, removal and motion in a scenario comprised of static and mobile nodes.

The following is a detailed description of ROME. In ROME each node stores two constant values, `nodeID` and `nodeType`, that indicate the node unique identifier and its type: M for mobile nodes and S for static ones. We stipulate that a mobile node detects motion by means of some function (like the *inMotion()* used below) based on some on board device, such as an accelerometer or the like. A node also maintains the two parameters `nodeCoord` and `nodeColor`. The first contains the coordinates of the node, which vary only if `nodeType` $= M$. Whether geodetic or virtual, these coordinates are needed only when nodes are not moving, and are computed by a node through specialized hardware (e.g., GPS) or by executing a localization protocol as soon as it stops. The parameter `nodeColor` takes values from C_0 to $2C_{max}$ and the value *undefined* that means that the node has no color. At the start of the network operations every node takes the color *undefined*. When a new node is added to the network or when a mobile node stops and needs to recompute its color, `nodeColor` is set instead to $C_{max}+1$. `nodeColor` takes values in the range $C_{max}+1$ to $2C_{max}$ only while nodes are trying to find a route to the sink. When the route is found, this parameter only takes values from C_0 to C_{max}. Finally, the sink is always awake, and it is colored C_0.

The protocol is described by the following procedures, where it is assumed that when a node is mobile it has no color. When a node needs to send a packet it executes the procedure `sendPacket` below (Algorithm 1).

Algorithm 1 procedure `sendPacket(dataPkt)`

```
 1: R, T ← NIL
 2: C ← undefined
 3: if freeChannel() then
 4:    findRelay(R, C, T);
 5: if (C ≠ undefined) then
 6:    transmitData(dataPkt, R)
 7:    if (T = S) and (not inMotion()) then
 8:       nodeColor ← C
 9: else
10:    if (nodeColor ≠ undefined) then
11:       if unsuitable(nodeColor) then
12:          nodeColor ← nodeColor + 1
13:    compute backoff
14:    if transmittable(dataPkt,backoff) then
15:       try sendPacket(dataPkt) after backoff
16:    else
17:       discard(dataPkt)
```

In this case the node needs to find a relay and if it is not moving it also needs to find a color. For finding them it initializes three variables, namely, R for the relay, T for the type, and C for the color, to NIL and *undefined*, respectively. If the channel is sensed free (function *freeChannel()*, line 1) the procedure `findRelay` (Algorithm 2) is executed to find suitable values for R, T and C, if any. If a relay R is found (this is the case when C is defined, line 1), the node transmits the packet to it (line 1). The procedure *transmitData* includes the reception of an acknowledgment for `dataPkt` (and provisions for later retransmission in case the sending is unsuccessful). Bursts of packets can be transmitted back to back. If the selected relay R is a static node, and if the executing node is not moving (checked through the function *inMotion()*, line 1), the executing node also updates its `nodeColor` variable to the one found by `findRelay` (line 1). Observe than a node keeps a color only if it does

978-1-4244-4148-8/09/$25.00 ©2009
This full text paper was peer reviewed at the direction of IEEE Communications Society subject matter experts for publication in the IEEE "GLOBECOM" 2009 proceedings.

not move; Otherwise, it leaves the nodeColor value to undefined. This is why we check that the executing node is not *inMotion()*. The fact that only static nodes update the color of an executing node is because of optimization reasons: It does not affect the protocol correctness and improves performance (see examples below). If no relay (and color) could be found by findRelay the **else** of line 1 is executed. If the node has a color it has to check whether this color is still suitable for forwarding packets. The current attempt of finding a relay might in fact have failed because of all potential relays were asleep. The function *unsuitable()* checks whether enough attempts have been made to find such relays by keeping and updating a counter for the current value of nodeColor. Specifically, a given number N_{att} of consecutive attempts are necessary to a node for deciding whether it has a relay of a feasible color, in the right direction or not. In case N_{att} unsuccessful attempts have been performed, the node tries to forward the packet in a different direction by changing its color to nodeColor $+1$, i.e., to the next possible color. The same number of attempts N_{att} will be devoted to this color. (If a node reaches C_{max} it stops incrementing and keeps this color.)

Every time the relay selection process has failed, which may happen because of a busy channel, a failure during the *transmitData* or when no relay was found, a random backoff time is computed (line 1). By executing the function *transmittable()* the node then checks if after the backoff it makes sense to reschedule the transmission of the packet. If it does, the sendPacket procedure is called again for packet dataPkt after backoff time has passed. Otherwise, the packet is discarded (line 1). The function *transmittable(dataPkt, backoff)* is completely general, and can be defined to consider a given number of possible retransmission at the MAC layer as well as the application-mandated lifetime of dataPkt in a cross-layer fashion.

The procedure findRelay is crucial for sending a packet. It is described by Algorithm 2.

Algorithm 2 procedure findRelay(r, c, t)

```
1: if (nodeColor = undefined) then
2:    scanColor(r, c, t, C_0, C_max)
3: else
4:    scanColor(r, c, t, C_0, nodeColor)
```

If a node has no color, it executes the procedure scanColor (Algorithm 3) to acquire one and to find, at the same time, an available relay. If the node is colorless, the color is sought for in the whole range of possible colors (line 2). The procedure scanColor always returns the minimum color (i.e., the best one) among those available in the specified range. In case the node has a color, it tries to upgrade its color to a better one, since some node could have arrived in the meanwhile that can be a better relay toward the sink. If no upgrade is possible, the node seeks for a relay compatible with nodeColor. That is why the procedure scanColor is called at this time on the range from C_0 to the nodeColor. The procedure scanColor(R, C, T, mc, MC) is described by Algorithm 3 below.

Algorithm 3 procedure scanColor(R, C, T, mc, MC)

```
1: broadcast(RTS)
2: wait for event for τ time
3: if event = CTS then
4:    R ← CTS.nodeID
5:    C ← CTS.senderColor
6:    T ← CTS.senderType
7: else if event = collision then
8:    if mc < MC then
9:       scanColor(R, C, T, mc, ⌊(mc + MC)/2⌋)
10:      if C = undefined then
11:         scanColor(R, C, T, ⌊(mc + MC)/2⌋ + 1, MC)
12:   else if mc = MC then
13:      binarySplitTree(R, C, mc)
```

It receives as input R and T initialized to NIL and C initialized to *undefined* by the procedure sendPacket. These parameters will contain a relay, its type and a color, respectively, if any are to be found. The procedure also takes two values mc and MC, $mc \leq MC$, that specify the range of colors that the executing node can assume. This range of colors determines the set of currently colored and awake neighbors that can candidate themselves as relays (feasible relays). The smallest the range, the smallest the cardinality of this set. The procedure scanColor is based on the CSMA-like, RTS/CTS-based mechanism for finding relays that is typical of cross-layer protocols for geographic forwarding [6]. The node broadcasts a request-to-send (RTS). The control packet RTS includes the following fields. The integers mc and MC that delimit the range of colors that the sending node may assume. The RTS also carries the sender nodeID, nodeType, nodeColor and nodeCoord as well as the coordinates of the sink. Upon receiving an RTS, feasible relays immediately answer each broadcasting a clear-to-send (CTS) packet containing their own ID and type, and the color that the sender should have to select that node as the relay (senderColor). After transmitting the RTS the sender listens to the channel for a specified amount of time τ. This is the time needed for the RTS to propagate and be processed at the receivers, and for CTSs to propagate back to the sender. Three events are possible. The CTS from one feasible relay is received correctly (line 3). In this case, the answering node will be selected as the forwarder of the packet. The parameters R, C and T are updated accordingly. The second possible event happens when multiple neighbors broadcast their CTS at the same time, i.e., a collision occurs. Since we want scanColor to return one of the relays with the smallest color (if any), we split the original interval (mc, MC) into two and proceed recursively to find a relay starting with the one that could induce a color for the sender in the lowest range of colors (line 3). If no relay can be found among these, the search proceeds among the nodes that could induce a color from the highest range (line 3). This leads us to find relays that would induce the smallest color for the sender (if any). To select among nodes with smaller color we use a binary split tree procedure (*binarySplitTree()*, line 3). Collision of multiple

978-1-4244-4148-8/09/$25.00 ©2009

nodes are resolved by having candidate relays flipping fair coins in successive rounds and broadcasting a CTS only if head occurs. If no node transmits in a round, the sender asks those that participated to the previous round to flip their coins again, until only one node successfully gets its CTS across. The third possible event is that no node replies to the RTS. In this case, the `scanColor` takes no action, and the control returns to the procedures `findRelay` and `sendPacket`.

Neighbors of the sender that are awake and colored when the sender itself broadcasts the RTS execute the following procedure `OnReceiving`(RTS) (Algorithm 4).

Algorithm 4 procedure `OnReceiving`(RTS)

```
1: if (nodeColor ≠ undefined) then
2:    senderColor ← getColor(RTS.nodeCoord)
3:    if senderColor ∈ [RTS.mc,RTS.MC] then
4:       CTS.nodeColor ← senderColor
5:       CTS.nodeID ← nodeID
6:       CTS.senderType ← nodeType
7:       broadcast(CTS)
```

The local variable `senderColor` is initialized by the function *getColor* that given the coordinates of the sender, and using the coordinates of the sink, those of the executing node and its color determines the color that the sender should have to use this node as a relay. Specifically, let x be the sender and y be the node executing the procedure `OnReceiving`(RTS). If y is in F of node x then `getColor` returns the smallest even color among those greater than or equal to the color of y. If instead y is in F^C of node x, then `getColor` returns the smallest odd color among those greater than or equal to the color of y. If `senderColor` belongs to the set of colors that the sender is looking to get, then this node is a candidate for relaying the packet, and broadcasts back a CTS (line 4).

We conclude this section with some examples aimed at showing situations successfully tackled by ROME.

Moving mobile nodes. While on the move a mobile node m has an undefined color. It therefore never offers itself as a relay for another node (the value undefined is never in the valid range for answering an RTS). When m has a packet to transmit it executes the procedure `sendPacket`. A relay is searched (Algorithm 2) in the range $C_0, \ldots, C_{max}$. Specifically, m selects as forwarder the eligible relay that would assign it the minimum color in this range. If such a relay exists m delivers the data packet to it, otherwise it backs off and tries again later. In any case its color is not updated (Algorithm 1, line 1) and stays undefined until the mobile node is on the move.

A mobile node stops or a new node is added to the network. In this case the node color is set to $C_{max} + 1$. Let x be the node. If x has static neighbors with a route to the sink, i.e., with a color in $C_0, \ldots, C_{max}$, they will answer x RTS with a CTS when the procedure `findRelay` is executed by x (Algorithm 2, line 2). As soon as a static node answers node x RTS, node x assumes the color c_x in $C_0, \ldots, C_{max}$(Algorithm 1, line 1). From then on, x participates to contentions initiated by nodes colored in $C_x, \ldots, C_{max}$ and is selected as relay if it provides a better color than other eligible relays. However, if node x is a mobile node which has stopped at a given location, its selection as relay by a node y will not change node y color (Algorithm 1, line 1). This rule is the result of several experiments and is an optimization we have adopted to improve performance. If we allowed static nodes to update their color based on mobile nodes, static nodes would assume colors smaller than the ones they can keep when the mobile nodes move away again. When such event happens, the static node will then have to increase its color. Such procedure requires the node to perform N_{att} transmission attempts before it realizes the `nodeColor` parameter has to be increased. It is therefore undesirable to perform this kind of operations very often. The current rule for changing color upon a relay selection solves this problem. Static nodes are allowed to use as relays colored mobile nodes, but maintain as color the one they would assume if they could only use routes going through static nodes.

A mobile node leaves a location. When a mobile node m starts moving its color is set to undefined, and the node acts as in the case of a moving mobile node. This is completely transparent to the colored static nodes in the area from which m left. Upon the mobile node departure these nodes still have a valid color (since the mobile node has never affected their coloring) and choose among the relays they were using before the mobile node arrival (if they did not change their color).

Nodes death or failure. It may occur that a static node x "dies" and is removed from the network. This case has no impact on network operations unless there are nodes that get disconnected or have to change their color because of x removal. If nodes get disconnected they will no longer be able to transmit packets until new nodes are added or arrive close by that re-connect them to the rest of the network with routes to the sink. The case when a node is no longer able to find relays of the proper color is handled in Algorithm 1, lines from 1 to 1. Each time the node acquires the channel and performs an unsuccessful contention to find a relay a counter is increased. Whenever the relay search has been unsuccessful for N_{att} consecutive attempts, i.e., no eligible relays in the right color range can answer, the node increases its color (line 1). This goes on until the node assumes a color which allows it to find feasible relays. From then on the node is able to correctly and quickly transmit its packets according to ROME operations.

III. Performance Evaluation

The efficiency and effectiveness of ROME in delivering packets to a static sink have been assessed through an extensive set of ns2 simulations on realistic scenarios. We consider networks made up of a number $N = 300$ of nodes randomly and uniformly scattered over a square area of side 320m. Of the N nodes, $p_s N$ of them are static and the remaining $p_m N$ are mobile, p_m, $p_s \leq 1$ and $p_m = 1 - p_s$. In our simulations we varied p_m in the set $\{0.2, 0.5\}$ These settings corresponds to scenarios where each static node has an average of 11 and 7 static neighbors, respectively. Each sensor node has a transmission range set to 40m. Nodes alternate between awake and asleep states according to a duty cycle $d = 0.1$. The

978-1-4244-4148-8/09/$25.00 ©2009
This full text paper was peer reviewed at the direction of IEEE Communications Society subject matter experts for publication in the IEEE "GLOBECOM" 2009 proceedings.

nodal energy model is that of the TelosB sensor nodes [9]. The channel data rate is $38,400$Kbps. Traffic is generated according to a Poisson process with parameter $\lambda = 1$, 2, and 4 packets/s. Once generated, a packet is assigned to a sensor node selected randomly and uniformly among the deployed nodes. Packets are addressed to a sink that is randomly placed in the deployment area. Varying the parameter λ allows us to test the protocol under varying traffic conditions, from moderate to high traffic.

We have performed simulations on various scenarios to evaluate the performance of ROME with mobile nodes, when new nodes are added to the network and when some nodes are removed. For lack of space the experiments described here concern only the case of a percentage of the network nodes being able to move through the network. We consider scenarios where $p_s N$ static nodes are first statically deployed. During the first 5000s of the simulation these nodes get their color (which requires a few hundreds of seconds) and exchange data according to ROME operations. After this time the mobile sensor nodes are added to the network, and start generating traffic. Their initial deployment is random and uniform, and then they are left free to roam according to the random waypoint mobility model. The maximum speed of a mobile sensor node is set to 5m/s, while the pause time p_t at a given location has been varied among the values in $\{60, 240, 900\}$s to investigate the impact of slow and aggressive mobility on the performance of ROME. Mobile and static sensor nodes are homogeneous, which imposes the same transceiver technology, the same communication-induced energy consumption, and the same buffer size, which is set to 20 packets. All results have been obtained by averaging over 50 different topologies. Each simulation run lasts for a total of $60,000$s.

We have investigated the following metrics: 1) Packet Delivery Ratio; 2) Average route length; 3) End-to-end packet latency; 2) Normalized energy consumption and 5) protocol overhead. The normalized energy consumption is defined as the energy consumed by the network per bit of information successfully delivered to the sink. The metric only considers the energy consumption imposed by communications (and not, for instance, by nodal motion). The protocol overhead (simply overhead in the following) is given by B_t/B_d, where B_t is the amount of traffic (control, data) transmitted in the network (in bits), and B_d is the data traffic successfully received at the sink (in bits). This metric expresses the amount of traffic transmitted in the network per bit of information delivered to the sink. The overhead is trivially bounded by 1 from below, and, in a multi-hop network, cannot be lower than the average number of hops traversed by the packets. In fact, we expect higher overhead values, since each protocol pays some overhead for header transmission, control packets exchange (including acknowledgments), and packet retransmissions in case of failed handshake.

Our investigation in this paper is based on two sets of experiments. The first experiment aims at assessing ROME superior performance over related solutions previously proposed for mobile ad hoc networks. The natural benchmark is Greedy Perimeter Stateless Routing (GPSR) [7]. Beyond being one of the most established solutions for geographic routing, GPSR combines mechanisms for dealing with dead-ends with mobility support. Dead-ends are dealt with through network topology planarization and face routing. To be able to correctly operate GPSR nodes thus need to have an updated local view of their neighborhood, which is based on periodic beacon exchange. In our experiments we have used the ns2 code of GPSR provided by the authors, which relies on standard CSMA/CA. With respect to the available implementation we have reduced the header, decreased RTS and CTS packet sizes, increased the slot time to 0.2ms and tuned the beacon period to 5s, all with the aim to maximize GPSR performance in our specific simulation scenario. We observed that increasing the beacon rate beyond 5s would increase the overhead without benefitting the protocol performance. Decreasing the beacon rate would instead make the neighborhood information stored at the nodes stale, which would impose significant packet loss. We have also implemented all the optimizations suggested by the authors to improve the performance of GPSR. Beyond transmitting periodic beacons, GPSR piggybacks beacon information in the transmitted data packets (this makes the beacon exchange more robust.) Exchange of the information needed to maintain face routes is performed reactively, since the original proactive solution would significantly increase the overhead and would not be a viable option for low data rates technologies. Finally, we activated the interface queue traversal option. Upon notification of a MAC retransmit retry failure, GPSR traverses the queue of packets for the interface, and removes all packets addressed to the failed transmission's recipient. These packets are sent back to the routing protocol for re-forwarding to a different next hop. All these optimization boost GPSR performance noticeably.

We have also benchmarked ROME vs. deterministic and probabilistic flooding. This allowed us to check ROME performance with respect to simple schemes that use a brute force approach (redundant packet transmission) to deal with nodes mobility. With respect to deterministic flooding, probabilistic flooding prunes some of the network transmissions trading-off reliability (i.e., the ability to reach all nodes) with decreased overhead and energy consumption. According to probabilistic flooding each node receiving a packet for the first time relays it further with probability p and discards it with probability $1 - p$. We have varied p from 0.5 to 1. (The latter value implements deterministic flooding.) Note that both GPSR and probabilistic flooding do not support nodes duty cycling, i.e., their performance decreases remarkably if the nodes follow a duty cycle. Differently from ROME, in GPSR and probabilistic flooding nodes are always awake.

Results of our comparative performance evaluation are shown in Table I and Table II. The displayed values refer to the time when both static and mobile nodes are active in the network. At the considered traffic loads ROME is able to deliver to the sink 100% of the generated packets. At the same traffic loads both the GPSR and flooding protocols get congested. GPSR suffers a 22% packet loss in scenarios with

978-1-4244-4148-8/09/$25.00 ©2009

Table I
ROME VS. GPSR, $p_m = 0.2$,$p_t = 240s$

λ	1.0		2.0		4.0	
Metrics ↓ | Protocols →	ROME	GPSR	ROME	GPSR	ROME	GPSR
Packet Delivery Ratio	1	0.98	1	0.78	1	0.32
End-to-end Latency (s)	5.42	1.97	7.29	7.7	17.71	29.2
Route Length (hops)	8.42	7.23	8.51	10.7	8.83	18.9
Normalized Energy Consumption(mJ/bit)	0.00029	0.00293	0.00015	0.00200	0.00009	0.00270
Overhead	13.75	15.13	13.5	44.05	13.33	139.5

Table II
FLOODING AND PROBABILISTIC FLOODING, $\lambda = 1$, $p_m = 0.2$, $p_t = 240$S

Metrics ↓ | Forwarding probability →	1.0	0.8	0.7
Packet Delivery Ratio	0.94	0.93	0.9
End-to-end Latency (s)	3.73	3.6	3.6
Route Length (hops)	9.2	9.1	9.0
Normalized Energy Consumption (mJ/bit)	0.0033	0.0033	0.0033
Overhead	303.3	244.4	215.7

Table III
ROME PERFORMANCE, $\lambda = 1$, $p_m = 0.2, 0.5$, $p_t = 60s, 90s$

p_m	0.2		0.5	
Metrics ↓ | Pause time p_t →	$60s$	$900s$	$60s$	$900s$
Packet Delivery Ratio	1	1	1	1
End-to-end Latency (s)	$5.62(s), 5.55(m)$	$4.83(s), 4.9(m)$	$11.25(s), 11.26(m)$	$7.9(s), 7.96(m)$
Route Length (hops)	$8.34(s), 8.48(m)$	$8.35(s), 8.57(m)$	$9.66(s), 10.02(m)$	$9.57(s), 9.87(m)$
Normalized Energy Consumption (mJ/bit)	0.00027	0.00030	0.00021	0.00029
Overhead	13.7	13.61	16.03	15.52

$\lambda = 2$ and is able to deliver only 32% of the generated packets when λ is set to 4. The network gets congested even earlier in case packets are flooded: A significant percentage of packets is lost already when $\lambda = 1$, especially when we use probabilistic flooding. In the latter case the unreliability of the scheme, which does not guarantee that all nodes will be reached by the flooded packet, makes packet delivery very challenging.

ROME has also much better performance in terms of energy consumption. Since the protocol has been carefully designed to allow nodes to follow a duty cycle (0.1 in our experiments) it is expected that it will have superior performance over GPSR and probabilistic flooding for what concerns the normalized energy consumption. However the gap is higher than what one would expect. This stems from ROME nodal energy consumption which only slightly increases with the traffic load, and, even at high traffic, is not significantly higher than the nominal duty cycle. The reasons for such good energy performance are multifold, and include: The transmissions required by the swift ROME relay selection operations, their being performed through control packets that impose low overhead, and the use of energy efficient techniques (e.g., allowing static nodes to go to sleep as soon as they realize they won't be selected as relay in the current contention, allowing nodes in motion to go to sleep if they are not handling packets). The possibility to transmit bursts of packets back-to-back when a relay is found (a feature which decreases the energy required to successfully relay a packet at high traffic load) justifies the fact that performance do not degrade with the traffic load. The fact that the nodal energy consumption does not significantly increase with traffic also motivates the significant improvements in terms of normalized energy consumption. At higher traffic the energy consumed by the network is exploited to successfully deliver a much higher amount of information, resulting in decreased normalized energy consumption.

In terms of latency we would expect GPSR and probabilistic flooding to outperform ROME. This should be the price to pay for improved energy performance. In ROME nodes alternate between awake and asleep modes, appearing and disappearing from the networks. This makes finding an awake relay more challenging in ROME than in schemes where nodes are always ON. It is therefore expected that GPSR and probabilistic flooding will experience much better latency performance. Results shown in Table I and Table II partially contradict this intuition. At moderately low traffic ($\lambda = 1$) GPSR and probabilistic flooding outperform ROME. The improvement is more limited in case of flooding (which has an average latency of 3.73s vs. the 5.42s experienced by ROME) and more significant in GPSR (whose average latency is around $1/3$ of that experienced by ROME). However, as the traffic increases the trend changes. When $\lambda = 4$, despite the high percentage of packet loss, GPSR experiences 65% higher latency than ROME (29.2s vs. 17.7s). The reason is twofold. The average length of the routes traversed by GPSR packets significantly increases with traffic (18.9 hops at $\lambda = 4$ vs. the 7.23 hops experienced by the same protocol at $\lambda = 1$). Route length is instead more or less constant in ROME. The overhead is also higher in GPSR, and significantly increases with traffic. In ROME the overhead instead slightly decreases

978-1-4244-4148-8/09/$25.00 ©2009

with traffic, thanks to the positive effect of back-to-back transmissions. Energy consumption and overhead results clearly show the limits of probabilistic flooding. Beyond suffering from packet loss this solution results in very high overhead (one order of magnitude higher than ROME), thus also in higher energy consumption. The protocol mostly pays for the redundant transmission of information. ROME overhead performance confirms the lightweight nature of our protocol. Results show that the amount of extra traffic transmitted per hop in addition to data payloads is limited to around 60% of the transmitted data. In our settings control packets are $1/10$ the data packets. The minimum amount of control packets needed by an RTS/CTS based protocol to advance one data packet is one RTS (to trigger CTS responses), one CTS (transmitted by the selected relay), and one ACK packet. The actual protocol overhead is not significantly higher showing that relay selection is fast, confirming that the number of attempts to find a relay is limited (around 1.5, on average), and that retransmissions have little or no impact on the overhead. Nodes do not pay in terms of overhead if they are unable to access the channel. Failed attempts to find any awake relay cost a single RTS transmission.

The second set of experiments has been devoted to evaluate the impact of aggressive mobility on ROME performance. We have varied the percentage of mobile nodes and the pause time.

Results are displayed in Table III for $p_t = \{60s, 900s\}$ and $p_m = \{0.2, 0.5\}$. The traffic load for these experiments has been set to $\lambda = 1$. Latency and route length values are labeled with either the identifier s or m, indicating whether the metric refers to packets generated by static or mobile sensor nodes. Our experiments show that ROME is able to cope well even with high nodal mobility. All generated packets are successfully delivered. More aggressive mobility (i.e., a higher number of mobile nodes or a shorter pause time) mostly results in increased latency. When the nodes are on the move they cannot serve as relays for other packets. Therefore aggressive mobility decreases the set of eligible relays of a given node, making more challenging to advance packets. It also reduces the number of nodes that share the large majority of the network load. This tends to congest nodes earlier. Normalized energy consumption instead decreases with mobility. In ROME nodes aggressively go to sleep whenever they know that they will not be involved in packets transmissions and receptions. This is the case of mobile node on the move that can go to sleep until they generate new packets. Scenarios with a higher number of mobile nodes tend to increase the load of static nodes. However, the energy consumption of mobile nodes is decreased. Mobile nodes relay their packets to the currently awake neighbor with the lowest color, which may not be the neighbor with minimum color. This motivates why the route traversed by packets generated by mobile nodes tend to be slightly longer than those traversed by packets generated by static nodes.

Finally, we have also run experiments to evaluate the performance of a network made of Np_s static nodes when $\lambda = 1$. Our aim here is that of showing that adding mobile nodes actually improves performance. Results show significant improvement in the hybrid case (static and mobile nodes) with respect to the totally static case. The gap is more significant when the pause time increases, enabling mobile nodes to contribute to packet forwarding for larger percentages of time. For instance, when $p_t = 900$s, $p_m = 0.5$ the latency experienced in the hybrid case is 34% lower than when only static nodes are present.

IV. Conclusions

In this paper we contributed to research in WSNs by defining and testing, ROME, a routing protocols for networks with mobile nodes. While dealing with mobility, ROME also retains desirable properties of protocols proposed for routing in static WSNs, which include dealing with node addition and removal as well as congestion and connectivity holes. Through simulations we show that on scenarios with static and mobile nodes together ROME is an efficient solution that delivers all generated packets to the sink and outperforms ad hoc schemes that have been proposed for mobile networks.

V. Acknowledgments

This work was partially supported by the FP7 EU project "SENSEI, Integrating the Physical with the Digital World of the Network of the Future," Grant Agreement Number 215923, www.ict-sensei.org, and by the EU ARTEMIS project #100017 "SOFIA, Smart Objects For Intelligent Applications."

References

[1] I. Demirkol, C. Ersoy, and F. Alagöz, "MAC protocols for wireless sensor networks: A survey," *IEEE Communications Magazine*, vol. 44, no. 4, pp. 115–121, April 2006.

[2] K. Akkaya and M. Younis, "A survey on routing protocols for wireless sensor networks," *Elsevier Ad Hoc Networks*, vol. 3, no. 3, pp. 325–349, May 2005.

[3] L. Zou, M. Lu, and Z. Xiong, "A distributed algorithm for the dead end problem of location based routing in sensor networks," *IEEE Transactions on Vehicular Technology*, vol. 54, no. 4, pp. 1509–1522, July 2005.

[4] B. Yu, P. Scerri, K. Sycara, Y. Xu, and M. Lewis, "Scalable and reliable data delivery in mobile ad hoc sensor networks," in *Proceedings of ACM AAMAS 2006*, Hokkaido, Japan, May 8–12 2006, pp. 1071–1078.

[5] B. Ren, J. Ma, and C. Chen, "The hybrid mobile wireless sensor networks for data gathering," in *Proceedings of the ACM International Conference on Wireless Communications and Mobile Computing, IWCMC 2006*, Vancouver, BC, Canada, July 3–6 2006, pp. 1085–1090.

[6] P. Casari, M. Nati, C. Petrioli, and M. Zorzi, "Efficient non-planar routing around dead ends in sparse topologies using random forwarding," in *Proceedings of the IEEE International Conference on Communications, ICC 2007*, Glasgow, Scotland, June 24–28 2007.

[7] B. K. and H. T. Kung, "GPSR: Greedy perimeter stateless routing for wireless networks," in *Proceedings of the* 6*th ACM Annual International Conference on Mobile Computing and Networking, MobiCom 2000*. Boston, Massachusetts: ACM, 2000, pp. 243–254.

[8] S. Basagni, M. Nati, and C. Petrioli, "Localization error-resilient geographic routing for wireless sensor networks," in *Proceedings of IEEE Globecom 2008*, New Orleans, LA, November 30–December 4 2008.

[9] J. Polastre, R. Szewczyk, and D. Culler, "Telos: Enabling ultra-low power wireless research," in *Proceedings of the* 4*th International Symposium on Information Processing in Sensor Networks, IPSN 2005*, Los Angeles, CA, April 25–27 2005, pp. 364–369.

978-1-4244-4148-8/09/$25.00 ©2009