An End-to-End routing protocol for Peer-to-Peer communication in Wireless Sensor Networks

Ricardo Simón Carbajo
carbajor@cs.tcd.ie

Meriel Huggard
Meriel.Huggard@cs.tcd.ie

Ciarán Mc Goldrick
Ciaran.McGoldrick@cs.tcd.ie

Department of Computer Science, Trinity College Dublin, Ireland

ABSTRACT

Interfacing Wireless Sensor Network (WSN) technologies with the Internet is a key requirement for making sensor data globally available. To this end, the authors have developed the TinyTorrents system; a peer-to-peer publishing and redundancy framework for the dissemination of sensor data in a reliable, redundant and self-consistent manner using torrent technology. TinyTorrents utilises a reactive routing protocol for WSNs which incorporates bidirectionality, reliability and generic communications modalities. In this paper the routing protocol, TinyHop, is presented. TinyHop creates on-demand routes, is self managed and works in an end-to-end fashion. Any node in the mobile environment can establish communications with, and retrieve data from, any contactable node at any time. Thus any node can function as a base station or sink. Mobile elements (e.g. data mules) or static gateways can interconnect from different parts of the network, thereby balancing the traffic load and helping avoid network partition. The protocol has been implemented in TinyOS 2.0.2 and simulated in TOSSIM.

Categories and Subject Descriptors

C.2.1 [**Computer-Communication Networks**]: Network Architecture and Design

General Terms

Algorithms, Performance, Reliability

Keywords

TinyTorrents, TinyHop, Wireless Sensor Network, WSN, Peer to Peer, P2P, Reactive Routing Protocol

1. INTRODUCTION

Wireless Sensor Network (WSN) technologies are driving many of the significant advances currently being made in the field of data communications. End-users of WSN technologies want their sensed data to be available in a timely, accurate and secure fashion. TinyTorrents [4] has been proposed by the authors for the reliable, redundant and distributed dissemination of WSN data across the Internet. TinyTorrents exploits existing peer-to-peer (P2P) ideologies, where data is encapsulated into torrents and spread over the network in a fault-tolerant manner. In the TinyTorrents system, data accessibility is federated and mediated according to both device capabilities and the prevailing needs of end-users. WSN node and network capabilities are characterised and published, thus allowing for the creation of end-user applications that can access, consume and aggregate data from multiple, diverse sources. The TinyTorrents infrastructure incorporates a high reliability WSN routing layer (TinyHop) which delivers the bi-directional data communication features required to support the TinyTorrents architecture. TinyHop presumes that any mote can act as a gateway. Mobile or static elements (like data mules [12]) which implement the TinyTorrents system can integrate with the network from any location, thereby helping to balance the traffic load and avoiding energy-based network partition. The new routing protocol provides on-demand connectivity by using local flooding techniques and it guarantees reliability through local repair mechanisms. It can work with static and mobile networks, is self managed and is memory efficient as it tends to minimize the memory needed to perform routing. The protocol has been implemented using TinyOS version 2.0.2 and validated both in TOSSIM [9] and in a live WSN testbed.

The next section of this paper provides an overview of existing WSN reactive routing protocols. These protocols are reviewed and classified in the context of the operational requirements of TinyTorrents. The TinyHop protocol that underpins TinyTorrents is then presented, and its functional integration described. The performance of the protocol is then evaluated and, finally, some conclusions and planned improvements are presented.

2. ROUTING PROTOCOLS IN WSN

Features such as node mobility, energy awareness, reliability and reactive/proactive approaches are of significance in WSN environments and must be incorporated when designing routing protocols. A variety of approaches have been employed to reduce the power consumption of both the motes [3] and the WSN as a whole, thereby helping avoid network partition [2].

On-Demand protocols, such as AODV (Ad-hoc On-Demand

Distance Vector) [10] and DSR (Dynamic Source Routing) [6], have been used for traditional wireless ad hoc networks. Their reactive approach is attractive as it avoids the energy-greedy, periodic beacons used by proactive protocols. TinyAODV [13] and NST-AODV (Not So Tiny AODV) [5] are widely referenced, lightweight implementations of reactive WSN protocols.

TinyAODV [13] was designed to implement AODV using a small footprint. The current version (release 3) provides for communication between any two nodes in the network. Route Response (RREP) messages are only generated by the destination node. The routing metric employed is the hop count. Route errors are generated and locally broadcast when a data message can no longer be sent over a given path. This circumstance is detected using Link Layer Notifications (LLN) which are not enabled by default. The packet which initiates the route discovery process is discarded.

NST-AODV (Not So Tiny AODV) [5] is a reactive routing protocol which improves on TinyAODV at the expense of an increase in memory storage requirements. NST-AODV is similar to the proposed 6LoWPAN Ad Hoc On-Demand Distance Vector Routing (LOAD) protocol [7]. Unlike TinyAODV, NST-AODV uses local repair and an LLN mechanism is enabled by default, thereby supporting dynamic topology networks. Routes can be repaired without source initiated Route Discovery. RREP packets can now be generated by nodes other than the destination, leading to the use of partial routes in the discovery process. The packet that triggers Route Discovery is not discarded (as in TinyAODV) and routes are used immediately after the path is established. When unsuccessful link layer transmission occurs, two retries are triggered. When a link failure is detected the packet is buffered and transmitted if a new route is found. The additional memory expense involved in NST-AODV is as a result of the maintenance of two FIFO queues, one to save packets in the discovery phase and the other to record output packets. NST-AODV uses hop count as the routing metric.

3. TINYHOP: PROTOCOL DESCRIPTION

TinyHop is a reactive routing protocol which aims to minimize the number of messages necessary to perform routing and thus avoids the expensive (in power terms) periodic beacon messages of proactive routing protocols [11]. In scenarios where zones are defined (clusters), TinyHop may be configured to perform zone routing by limiting the flooding to the local area (hops or zone/group identifier). TinyHop is reliable in the sense that it provides bidirectional communication between any two motes in the network - if such communication is possible. It creates bi-directional routing paths at each discovery process thereby helping to minimise network partition. It allows any mote to act as an on-demand base station (sink) at any point in time.

In Table 1 TinyAODV, NST-AODV and TinyHop are compared (some data is drawn from [5]). TinyHop uses the fastest RREQ packet to establish the path, assuming that the route created will be both hop and congestion efficient. NST-AODV improves the TinyAODV protocol by introducing a local repair mechanism in which routes can be repaired without source-initiated Route Discovery. TinyHop implements a local repair mechanism which looks for alternative routes in the reverse phase of the Route Discovery process; this mechanism is employed to guarantee bidirectionality within the end-to-end route.

Table 1: Comparison of the features of TinyAODV, NST-AODV and TinyHop

Protocol	**TinyAODV**	**NST-AODV**	**TinyHop**
Connectivity Maintenance Mechanism	LLN (disabled)	LLN	End-to-End
RERR message	Yes	Yes	No
Local Repair	No	Yes	Yes
Only destination generates RREP	Yes	No	Yes
Routing Metric	Hop Count	Hop Count	Fastest RREQ
Precursor List	No	No	No
Implementation Status	TinyOS 1.x	TinyOS 1.x	TinyOS 2.x

TinyHop uses a flooding mechanism which seeds each node with information from the immediately previous node en route from the origin. This helps avoid cycles. As multiple paths are not maintained in every node, memory and power consumption overheads are reduced. In forward route discovery, metrics such as signal strength, number of hops, battery level and congestion level can be utilised in deciding which packet should be broadcast, although by default the metric used is based on the fastest RREQ to the destination. When the return route fails, local flooding is performed. This process exploits existing neighbourhood route information. The most suitable, valid neighbour route is selected (by default this is determined from the fastest ACK in response to the local broadcast).

TinyHop uses a flat, non-hierarchical network structure where any node may act as router, sink or source. Each end-to-end route between two nodes (source and destination) is independently discovered. Every packet from a source node is acknowledged by the destination node. This controls the path flow state and ensures that only fresh, working paths are maintained within the routing tables. Explicit acknowledgements, incorporating a dynamic backoff mechanism based on route reliability, are employed. Implicit acknowledgements are used on reliable, well established routes. Sequence numbers ensure that every packet is unique within a sequence cycle and facilitate discrimination between packets with the same ID that belong to different sequence cycles.

Every mote maintains two main tables: i) the Contactable Motes Table; a list of motes for which a route has already been created that is utilised for sending packets to the neighbour mote address with a specified packet sequence, and ii) the Routing Table; used to perform routing for each path (defined by its origin mote and destination mote). Every unique table entry is defined by a receive address and sequence (indicating where packets are received from), and a send address and sequence (indicating where the packets are to be sent). The acknowledgement packets traverse the route back to the origin mote. There is also a control field which acts as a reset counter to indicate activity on the path.

The protocol uses the following packet types: *RREQ*: a route is being requested; *RREP*: a response with route information is being sent back to the source; *A_RREP*: an acknowledgement that the *RREP* message was received by the next mote en route to the source; *LD_RREP*: when bidirectionality fails, a local discovery is performed to find new routes back to the source; *ALD_RREP*: an acknowledgement to the *LD_RREP* message, confirming that the packet has arrived at the next hop, that a bidirectional route exists and that there is a route back to the source; *RT*: the packet is traversing an existing route to the destination; *A_RT*: an acknowledgement packet for the RT message, traversing the bidirectional route to source and confirming that the message arrived and that the route is functional.

3.1 Protocol Phases

There are three distinct operational phases in the Tiny Hop protocol: the Route Discovery, Route Traversal and Route Maintenance phases.

3.1.1 Route Discovery Phase

Flooding has been employed in this protocol phase because of its inherent reliability. To minimize the number of packets involved in the flooding process, the protocol maintains a short dynamic list (which acts as a filter) containing the most recent packets received. This avoids duplicate retransmissions during the same flooding discovery process.

The Route Discovery phase is entered when a mote has a packet to transmit and there is i) no route to the destination in its "Contactable Motes" table, or ii) this route is no longer working. The packet "RREQ" contains two sequence numbers, one which acts as an identifier for the new route being created, while the other is modified at each hop. The mote sequence counter is increased for each new packet created by that mote. TinyHop currently operates on the basis of accepting the first arriving message and discarding subsequent requests.

If the "RREQ" packet is not in the list of recently received messages, a unique entry is created in the "Routing" table. This new entry contains the address of the sender and the sequence of the message, amongst other information. This phase of the protocol works by linking addresses and sequences from the motes wishing to route through the routing table mote. As packets flood through the network, information on how to reach the original source node is gleaned from them. This will be used in the acknowledgement process, completing the "Routing" table information.

Once the first "RREQ" is received by the destination mote, additional messages for the same route discovery process are discarded. An "RREP" acknowledgement packet is sent to the mote from which the first "RREQ" was received and contains the sequence information from that first "RREQ". The mote receiving the "RREP" packet will then identify the appropriate entry in the "Routing" table to route back the packet.

Each mote receiving an "RREP" packet must reply with an "A_RREP" packet, informing it that the packet has arrived successfully. Thus the "RREP" message will traverse the route back to the source to acknowledge the original "RREQ" message and confirm bidirectionality of the route. When the "RREP" message arrives at the source mote, the "Contactable Motes" table is updated.

For every "RREP" message sent back to the original source mote, an "A_RREP" acknowledgment message is expected by each intermediary node. If this is not received within a timeout period, the router mote assumes the packet was not delivered and therefore bidirectionality does not exist. In this case, a neighbourhood discovery process is launched to identify an alternative route to the original source mote. The mote broadcasts a message typed "LD_RREP". Neighbours reply with a message typed "ALD_RREP". The mote which started the "LD_RREP" process uses the first message received and discards the rest. In the "Routing" table, the original forward route (source-destination) is modified to connect with the new route. At this stage, the "RREP" process is restarted.

A difficulty arises when "A_RREP" packets are lost but the corresponding 'RREP" packet has been delivered successfully. In this scenario, a local route discovery process is launched which creates a different route from the one that is assumed to be broken. In this situation, when the source mote receives the "RREP", it will presume that the route information is valid. It will not be aware that a local route discovery process is underway, thus making the information in the "Routing" table invalid. This issue may be addressed by either maintaining information in the "Routing" table for all motes to which "RREP" packets have been sent, or through snooping packets once an "RREP" message has been sent. The latter approach reduces the number of messages involved in the discovery process, but at the expense of some additional computational cost and the interception of packets during the snooping operation. Both approaches are valid; indeed together they create a more reliable protocol.

3.1.2 Route Traversal Phase

This phase is entered when a source mote has data to send and there is an entry in the "Contactable Motes" table for the destination. The routing mote searches its routing table for the address and sequence number associated with the next node on the route. It updates the packet to be routed with this information and transmits it. Once the "RT" packet reaches the destination mote, an "A_RT" acknowledge packet will traverse the route back to the source mote.

3.1.3 Maintenance Phase

This phase of the TinyHop protocol is used to keep the number of entries in the "Routing" table within acceptable limits. In order to optimise network performance, it is essential to delete i) all entries created in the discovery phase that have never been used, and ii) valid routes that are no longer in use. A "Freq" field in the "Routing" table is set to 0 when a new entry is created during the route discovery phase. Every time an "RREP" or "A_RT" message is received it is incremented. If overflow of a mote's "Routing" table is imminent, a "cleaning" process is activated ensuring that only frequently used routes are maintained.

4. EVALUATION

The protocol has been developed on version 2.0.2 of the TinyOS platform [1]. The simulations presented were carried out using Tossim [9, 8], a TinyOS discrete-event simulator. TOSSIM simulates the micaZ motes (which utilising the CC2420 transceiver). The Radio Propagation model employed in TOSSIM simulates the RF noise and interference a node experiences. Signal-Noise Ratio (SNR) and Packet Reception Rate (PRR) responses, derived from experimental data, capture interference and noise transmissions originating both within, and external to, the WSN. Behaviours incorporated include those arising from RF shielding (utilising a variable attenuator [8]), interference bursts and other correlated problems. The model also uses the Closest Pattern Matching (CPM) algorithm to incorporate a statistical noise behaviour derived from a noise trace (Meyer) [8]. The Radio Propagation model ensures that the TOSSIM simulations appropriately capture many of the phenomena and effects that are experienced in a live WSN [8]. Local mobility and interference effects are implicitly incorporated in the

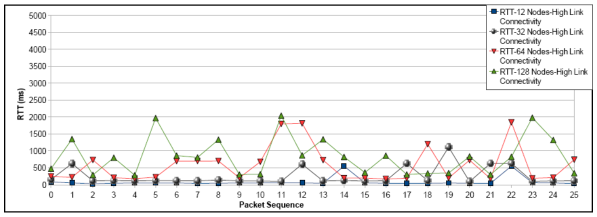

Figure 1: RTT per Packet Sequence Number for a scalable topology (12,34,64,128) with High Link Connectivity

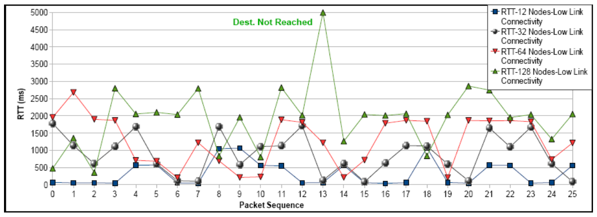

Figure 2: RTT per Packet Sequence Number for a scalable topology (12,34,64,128) with Low Link Connectivity

model's behaviour.

Two categories were defined to denote node connectivity scenarios: i) High Link Connectivity (-40dBm gain between nodes) and ii) Low Link Connectivity (-70dBm). In both scenarios the effect of the RF model ensures that packets get lost, thereby allowing for TinyHop's local repair behaviours to be executed.

Network topologies have been created for simulation and evaluation purposes. Nodes are randomly positioned in each topology and incorporate local mobility. Four different network configurations have been generated. The smallest is a 12 node topology, with end-to-end routes typically 3-4 hops in length. Intermediate configurations of 32 nodes (6-8 hops) and 64 nodes (8-15 hops) culminate in a 128 node configuration exhibiting end-to-end routes from 9 to 23 hops in length. In each topology paths are established according to scalability, with the origin node identified as node 1 and the destination node the most remote node in each topology (12,32,64,128).

The metric used to evaluate the different phases of the protocol was the RTT (round trip time). Here RTT is the time taken from when a packet with a specific sequence number is sent by a given user to when the user receives an acknowledgement of its successful reception at the destination node. For each user packet to be delivered successfully, TinyHop may need to perform more than one phase of the protocol. The RTT for a given packet is therefore the cumulative time taken in each of the different phases in the protocol for the successful transmission and acknowledgment of that packet.

For Figures 1 and 2 the New Discovery(ND) and Route Traversal(RT) protocol phases are artificially emphasised by setting their retransmission times to 800ms and 500ms respectively. These parameters are usually governed by an exponential backoff technique.

Simulation run length is set to 500 seconds, with 1 user packet passed into the TinyHop routing layer every 5 seconds. Each transmitted packet is 35 bytes long and TOSSIM determines the interhop delay based on the CC2420 transceiver data transfer rate (250Kbps) and the size of the packet.

The results presented in Figures 1 and 2 explore the scalability performance of the TinyHop protocol. Figure 1's results are generated using high link connectivity configurations (i.e. strong internode radio signals) whilst Figure 2's results arise from low link connectivity scenarios (i.e. weaker internode radio signals).

A careful examination of the RTT in every phase of the protocol shows how scalability affects the route discovery phase and local repair latency. The different phases of the TinyHop protocol are: ND (New Discovery) represents a new route discovery phase, LR (Local Repair) identifies that there has been a local repair within the route discovery phase, RT (Route Traversal) is when a packet is transmitted using a path and an acknowledgement is received, RTF (Route Traversal Failure) identifies that an RT packet has been resent as an acknowledgement was not received, RTM (Maximum Route Traversal Failure) indicates that the maximum attempts to resend an RT packet has been reached and NDM (New Discovery after Maximum RTF) is when a new route discovery is launched to find an alternative route.

In Figures 1 and 2, it can be seen that the RTT falls within determinable ranges. As expected the average RTT increases as longer paths are created with increases in topology sizes. Figures 1 and 2 demonstrate how the level of link connectivity impacts on connection reliability and protocol performance. For instance, it is clear that TinyHop offers better performance in low mobility, or near static, configurations. This is as expected from a reactive protocol. In low link connectivity scenarios (Figure 2), in which greater mobility is present, TinyHop executes more protocol phases to deliver a packet (e.g. RT retransmissions, new discovery process and local repair). Topological scalability mainly affects the reliability of the end-to-end communication - the RTT level in the 128 nodes topology is higher as the path is longer and more retransmissions are employed.

The protocol overhead when sending and receiving has also been evaluated. In Table 2 the same path, with origin set to node 1 and destination set to node 12, is discovered for all different scalable scenarios (12,32,64,128). This table highlights the overhead amplification factor that results from increasing the number of nodes in the evaluation scenario. Table 3 summarises the overhead information associated with Figures 1 and 2. In these configurations the origin is always node 1 and the destination is the most distant node in each of the scenarios (12,32,64,128). Finally, the impact of multiple distinct communications simultaneously traversing different paths in a 64 nodes scenario with low link connectivity is shown in Table 4. Comparing data from Tables 2 and 3, it is clear how scalability and link connectivity directly affect the overhead accruing to both received and sent packet counters. The topological scenario, link connectivity and discovered path length proportionately affect the number of broadcast and unicast packets lost. In low connectivity, or high local mobility, scenarios, routes may not exist for periods of time and the destination node may be unreachable (see Table 3, 128/L column). On the other hand longer paths (see Table 3 with higher number of hops) increase the number of packets received by the destination (RT received) as retransmissions and alternate route

Table 2: Topology Scalability: predetermined origin and destination nodes (1->12)

Nodes/Connectivity(High|Low)	**12/H**	**12/L**	**32/H**	**32/L**	**64/H**	**64/L**	**128/H**	**128/L**
Overhead Sent	918	1030	892	1279	934	1056	1193	1704
Overhead Received	3475	4293	4814	5703	4195	5542	5127	6923
Lost Packets(Broad/Unicast)	5b/23u	34b/45u	26b/19u	208b/58u	55b/20u	62b/50u	193b/23u	580b/57u
Avg. Num. Hops	4	4	4	5	4	4	4	5
Num. RT send	99	99	99	98	99	99	99	99
Num. RT received	107	117	104	112	105	116	111	125
Num. RT ack	98	96	99	92	99	99	98	99
Max. Routing Table Size	2	4	1	8	1	1	2	5

Table 3: Topology Scalability: different paths: 12 Nodes (1->12), 32 Nodes (1->32), 64 Nodes (1->64), 128 Nodes (1->128)

Nodes/Connectivity(High|Low)	**12/H**	**12/L**	**32/H**	**32/L**	**64/H**	**64/L**	**128/H**	**128/L**
Overhead Sent	919	1030	2175	3480	4564	9728	12551	22663
Overhead Received	3475	4293	8477	13280	17076	33292	45118	73805
Lost Packets(Broad/Unicast)	5b/23u	34b/45u	111b/45u	933b/163u	405b/78u	3946b/296u	3551b/174u	12548b/535u
Avg. Num. Hops	4	4	8	8	15	15	25	23
Num. RT sent	99	99	99	99	99	96	99	89
Num. RT received	107	117	117	121	118	99	127	66
Num. RT ack	98	96	95	24	93	54	71	21
Max. Routing Table Size	2	4	5	31	8	>50	35	>50

discoveries occur more often for the same packet sequence. The "RT ack" metric gives an insight into the number of user sequence packets successfully delivered. "RT sent" measures the number of user packets sent. Thus the number of route discoveries launched can be gauged by the difference between the two numbers. These discovery processes mainly arise from link connectivity issues along the route, but are also influenced by the number of hops which affects the delivery ratio and the number of route discovery phases needed (see Table 3, columns 64/L and 128/L). Contrasting the different link connectivity scenarios, it can be concluded that TinyHop is more sensitive (in terms of performance) to connectivity than path length. This is further evident in the number of routing tables entries, which increase with lower connectivity as more route discovery phases are performed. As expected, multiple paths created at the same time (see Table 4) also increase the overhead, in addition to the number of routing entries in the nodes.

5. CONCLUSIONS

TinyHop, a reactive routing protocol for end-to-end communication between wireless sensor network nodes, is presented. The protocol is purposed for applications employing a peer-to-peer paradigm but has much broader applicability. TinyHop seeks to minimize overheads in the communication process while offering reliability of packet delivery in a Peer-to-Peer environment. It utilises a local repair mechanism to achieve bidirectionality within the route discovery process. TinyHop has been implemented in TinyOS 2.x for micaZ motes (utilising the CC2420 radio transceiver). TinyHop is currently being extended to utilise a broader range of routing metrics, such as power consumption and link quality, and is being deployed on a live testbed.

Acknowledgement

This publication has emanated from research conducted with the financial support of Science Foundation Ireland.

Table 4: Topology Scalability: multiple origin and destination nodes: multiple paths.

Multipath (High Connectivity,64 Nodes)	**1 Path (2->59)**	**2 Path (2->59)|(26->54)**	**3 Path (2->59)|(26-54)|(19-44)**
Overhead Sent	3235	4584	7845
Overhead Received	13897	19855	31540
Lost Packets (Broad/Unicast)	333b/57u	325b/82u	1070b/176u
Avg. Num. Hops	11	11|7	11|8|7
Num. RT sent	99	99|99	99|99|99
Num. RT received	113	117|114	118|107|126
Num. RT ack	93	94|99	88|95|97
Max. Routing Table Size	8	7	22

6. REFERENCES

[1] TinyOS. Alliance TinyOS version 2.x, December 2007. http://www.tinyos.net/tinyos-2.x/.

[2] J. Chang and L. Tassiulas. Maximum lifetime routing in wireless sensor networks. *IEEE ACM T Network*, 12(4):609–619, 2004.

[3] J. Xu, D. A. Sumy, B. Vojcic. An overview of routing protocols for Mobile Ad Hoc Networks. In *Ultra Wideband Wireless Communication*, 341–427. 2006.

[4] K. Fritsche. TinyTorrent: Combining BitTorrent and SensorNets. *The University of Dublin, Trinity College, Dublin, Ireland, Tech. Rep. TCD-CS-2005-74,* 2005.

[5] C. Gomez, P. Salvatella, O. Alonso, and J. Paradells. Adapting AODV for IEEE 802.15.4 Mesh Sensor Networks: Theoretical discussion and performance evaluation in a real environment. In *WOWMOM '06: Proc. 2006 Int. Symp. on World of Wireless, Mobile and Multimedia Networks*, 159–170, Washington, DC, USA, 2006. IEEE Computer Society.

[6] D. Johnson and D. Maltz. Dynamic Source Routing in Ad Hoc Wireless Networks. *Mobile Computing*, 353(153-181):152, 1996.

[7] K. Kim, S. Park, G. Montenegro, and S. Yoo. 6LoWPAN Ad Hoc On-Demand Distance Vector Routing (LOAD). *draft-daniel-6lowpan-load-adhoc-routing-01, IETF*, 7, 2005.

[8] H. Lee, A. Cerpa, and P. Levis. Improving Wireless Simulation through Noise Modeling. *Proc. of the 6th Int. Conf. on Information Processing in Sensor Networks*, 21–30, 2007.

[9] P. Levis, N. Lee, M. Welsh, and D. Culler. TOSSIM: Accurate and Scalable Simulation of entire TinyOS Applications. *Proc. 1st Int. Conf. on Embedded Networked Sensor Systems*, 126–137, 2003.

[10] C. Perkins and E. Royer. Ad-Hoc On-demand Distance Vector Routing. *Proc. of the 2nd IEEE Workshop on Mobile Computing Systems and Applications*, 2:90–100, 1999.

[11] V. Raghunathan and C. Srivastava. Energy-aware Wireless Microsensor Networks. *IEEE Signal Proc Mag*, 19(2):40–50, 2002.

[12] R. Shah, S. Roy, S. Jain, and W. Brunette. Data MULEs: Modeling and analysis of a three-tier architecture for sparse sensor networks. *Ad Hoc Networks*, 1(2-3):215–233, 2003.

[13] Tymo source code repository. Tymo: DYMO implementation for TinyOS, December 2007. http://tymo.sourceforge.net.