Supporting Mobile Swarm Robotics in Low Power and Lossy Sensor Networks

Kevin Andrea
kandrea@masonlive.gmu.edu

Robert Simon
simon@gmu.edu

Sean Luke
sean@cs.gmu.edu

Department of Computer Science
George Mason University
4400 University Drive MSN 4A5
Fairfax, VA 22030 USA

Summary Wireless low-power and lossy networks (LLNs) are a key enabling technology for the deployment of massively scaled self-organizing sensor swarm systems. Supporting applications such as providing human users situational awareness across large areas requires that swarm-friendly LLNs effectively support communication between embedded and mobile devices, such as autonomous robots. The reason for this is that large scale embedded sensor applications such as unattended ground sensing systems typically do not have full end-to-end connectivity, but suffer frequent communication partitions. Further, it is desirable for many tactical applications to offload tasks to mobile robots. Despite the importance of support this communication pattern, there has been relatively little work in designing and evaluating LLN-friendly protocols capable of supporting such interactions.

This paper addresses the above problem by describing the design, implementation, and evaluation of the MoRoMi system. MoRoMi stands for Mobile Robotic MultI-sink. It is intended to support autonomous mobile robots that interact with embedded sensor swarms engaged in activities such as cooperative target observation, collective map building, and ant foraging. These activities benefit if the swarm can dynamically interact with embedded sensing and actuator systems that provide both local environmental or positional information and an ad-hoc communication system. Our paper quantifies the performance levels that can be expected using current swarm and LLN technologies.

1 Introduction

There is currently significant interest in the use of low cost and commodity-level robotic systems capable of performing collaborative actions using autonomous "swarm"-style intelligence. Additionally, there also has been an explosion in the use of wireless low-power and lossy networking (LLN) technology to support distributed autonomous environmental sensing applications, building and home automation systems, and the Internet-of-things. Although the

desirability of communication and interoperation between mobile swarms of autonomous robotics and embedded LLN sensing and communication systems has long been recognized, there has been relatively little work in designing and evaluating LLN-friendly protocols capable of supporting such interactions.

This paper addresses the above problem by describing the design, implementation, and evaluation of the *MoRoMi* system. MoRoMi stands for *Mo*bile *Ro*botic *M*ulti-sink. It is intended to support swarms of autonomous mobile robots engaged in activities such as cooperative target observation, collective map building, and ant foraging. These activities benefit if the swarm can dynamically interact with embedded sensing and actuator systems that provide both local environmental or positional information and an ad-hoc communication system [1]. Our paper quantifies the performance levels that can be expected using current swarm and LLN technologies.

MoRoMi is architected as an intentionally thin wrapper layer operating both within an open swarm robotics architecture [2] and over RPL, one of the standard LLN communication routing protocols [3]. The purpose of this design is to maintain compliance with evolving robotic and networking standards and to offer designers and researchers a simple tool to develop interoperable applications. MoRoMi supports multi-hop routing and communication between mobile robot swarms and embedded LLN systems. One primary goal is to rapidly form routing and forwarding ad hoc communication trees as robots enter and leave an LLN system *without* disrupting any existing LLN communication patterns or reporting functions. While others have described interactions between mobile robots and wireless LLN nodes [4], we are unaware of any other work that focuses on simultaneous support for both swarm robotic communication and preexisting LLN routing.

One of the greatest challenges we have faced is in maintaining frequently reconfigured routing trees on resource-constrained LLN nodes as mobile robots traversed through the system. Our objective is to minimize the time to form a new routing tree while maintaining acceptable levels of throughput through the system and simultaneously supporting preexisting LLN communication patterns. Our evaluation has shown that under common swarm communication requirements MoRoMi is an effective approach for swarm robotic to LLN communication.

2 MoRoMi Design

We first describe our robot platform, then discuss issues in LLN implementation

2.1 The Flockbots

The Flockbots are an open robot architecture designed to pack a large amount of sensors and useful effectors into as inexpensive a package as possible, so as to make it possible to build many copies of the robot. The platform also tries to use largely off-the-shelf equipment to facilitate this. The current Flockbot design is a differential drive design which uses an Arduino Uno microcontroller, which handles most sensor and effector tasks, attached to a Raspberry Pi Linux computer, which maintains a camera, 802.11 wireless communication, and USB. The Flockbot's basic facilities include five Sharp IR infrared range sensors, two bump sensors, two encoded wheels, an embedded gripper capable of collecting small cans, a surface for pushing boxes, and a tilt-servoed camera, and I2C-driven display; it can run for many hours on NiMH batteries. To this design we have attached a Tmote Sky sensor mote attached to the Raspberry Pi over USB.

Communications are facilitated by use of a socket API which allows for either processing by the main processing system or for direct control of the hardware by redirecting the traffic through to the Arduino board. The autonomous robots may be programmed either directly through dynamic C libraries and Lua scripts on-board the Raspberry Pi, or may be controlled remotely via the socket API interface.

As most computational tasks are performed on the Raspberry Pi, we can use on the sensor motes a streamlined version of Contiki OS, the operating system designed for LLNs [5]. This enables the Flockbot to load and interact with LLN node devices directly, using termios system calls to coordinate communications with the sensor mote. Using our API implemented on both the robot and the LLN nodes, we are able to not only program these devices on-board during normal operations, but also control the various aspects of LLN communications, to include transmission power levels and message sending rates to each destination. The Figure at right shows a Flockbot outfitted with the sensor mote.

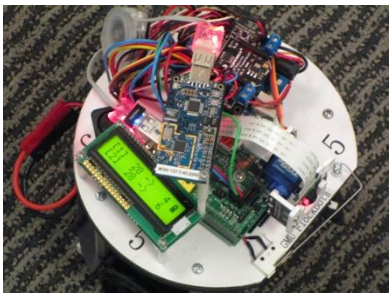

Figure 1: A FlockBot

2.2 Wireless LLN Node Implementation

Our targeted class of wireless LLN applications typically have a multipoint-to-point communication pattern, in which nodes collect or process information, and send their own packets or relay the packets from others up a tree to a common sink. In an LLN wireless environment, highly resource constrained systems generally use sink-oriented gradient routing techniques to maintain this tree [6]. Each node continually calculates the cost towards the sink, and forwards its packets along the cheapest gradient. The primary gradient protocol used in current systems is the IPv6-based Routing Protocol for Low-Power and Lossy Networks (RPL) [3].

RPL gradients take the form of Directed Acyclic Graphs (DAGs). Each DAG instance can be specified by its sink node, the set of basic metrics collected by each node about the quality of the link, a per-DAG method to combine these metrics into a single link metric, and a per-DAG method for determining path costs. Sinks are advertised to the network through the use of a Destination Oriented DAG Information Object, or DIO. Recent research has described some of the difficulties in using RPL under hostile but common LLN conditions. For instance, in [7] the authors evaluate the performance of RPL in a large testbed using both stable and node-failure scenarios. They observed that the path forwarding topology took a long time to stabilize and required a large amount of IPv6 control packet overhead.

Our approach in MoRoMi is to support multiple DAGs by establishing, on demand, gradients towards each mobile sink. As each sink enters the environment, it transmits a DIO to announce an Instance of the RPL network. Each of these Instances provide their own DAG to be formed. These DIO messages are broadcast wirelessly and are received by nodes within its transmission radius. Once received, each node will add information on this new DAG and rebroadcast the DIO using a higher rank, which is used to denote its own relative distance from the sink. This process will continue to form the initial DAG as well as periodically to ensure both the integrity of the DAG, and to enable nodes to change their logical parents as network conditions change. As a robot moves the gradient will shift reflecting its new position in the network, causing the DAG to adjust to ensure that communications are maintained with each respective sink.

As in [7] we found that stabilization was highly sensitive to beaconing and retransmission intervals. The trickle timer, as described by RFC 6550 [3] and implemented in Contiki, governs retransmission intervals of these DIO messages. This timer reduces activity by doubling the time between each message is sent, up to a default maximum interval of 2.3 hours between each message sent. One consequence of this is that a dynamic DAG consisting of many node-connected robots may suffer from delays in reforming. As a rapidly moving robot drives through its own DAG, each node receiving a DIO direct from the mobile sink will tend to accept it as a preferred parent, leaving a large DAG heavily disturbed and partitioned.

MoRoMi addresses this problem within the Contiki RPL implementation by adding an additional step during the forwarding process. When a MoRoMi node receives a message, it first examines the packet for routing information. In the most common occurrence, this message is destined for a distant sink and is immediately retransmitted to the current parent of the node for the appropriate DAG Instance, as indicated by the RPL option headers of the packet. During this routing process, the node reports the rerouting action and the message payload to the host robot, enabling forward dissemination of data upwards as it travels to the root of the DAG. Furthermore, MoRoMi reports each change in the preferred parent of a node in order to give the host information as to its own relative place in the DAG. Frequent parent changes, for instance, may indicate that the DAG is undergoing an adjustment, and MoRoMi tailors its routing message update rate to achieve rapid re-convergence.

3 Evaluation

Our evaluation has focused on determining the amount of time required for a newly introduced static robot to establish a new gradient routing path without disrupting pre-existing routing paths, gauging the ability of MoRoMi in support routing to mobile robotic sinks as the routing trees are reformed, and understanding both the velocity of the sink and the throughput of the system. For all results shown, all pre-existing communication patterns were maintained with no tree disruption and minimal variation in packet delivery rate.

3.1 Experimental Results

We conducted a number of experiments using the Flockbots and Tmote Sky LLN nodes. We present the results from one particular set of experiments, which are typical of the other results. We created a linear topology of seven Flockbots, with one Flockbot at each end acting as a sink. We then simulated a hierarchical swarm environment, where each of the sender robots generate and send a message to each of the two sinks in the environment. As each sink came online, it sent a DIO to the environment announcing its presence. Each Instance was used to identify and form its own DAG. After an initialization period one of the two sink Flockbots was driven from its initial location to a position on the other side of the perimeter. This represented a swarm leader performing a data gathering mission over the network. During mobility, the DAG of the mobile sink was disrupted as each node attempted to reassess and select the sink as a direct parent. As the robot was only in range of each node for a short duration, this parent information became stale and each node was forced to wait for a new DIO to be received before adjusting to the proper parent for the new configuration.

The mobile robot sink began at one end of the linear topology, then drove to the other end and dwelled for a specified period before returning. We observed eight states or phases during

operation. These phases were (1) the initial time after activation for the network to form, (2) the initial configuration operation, (3) the movement to the dwell location, (4) the time to reform at the dwell location, (5) the post-convergence operation at the dwell location, (6) the movement back to the initial position, (7) the time to reform at the initial location, and (8) the post-convergence operation at the initial location.

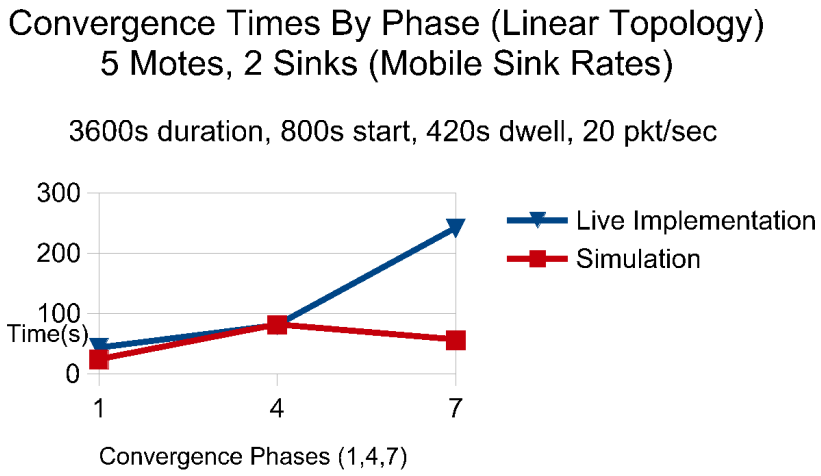

Figure 2: Convergence Times for Implementation and Simulator

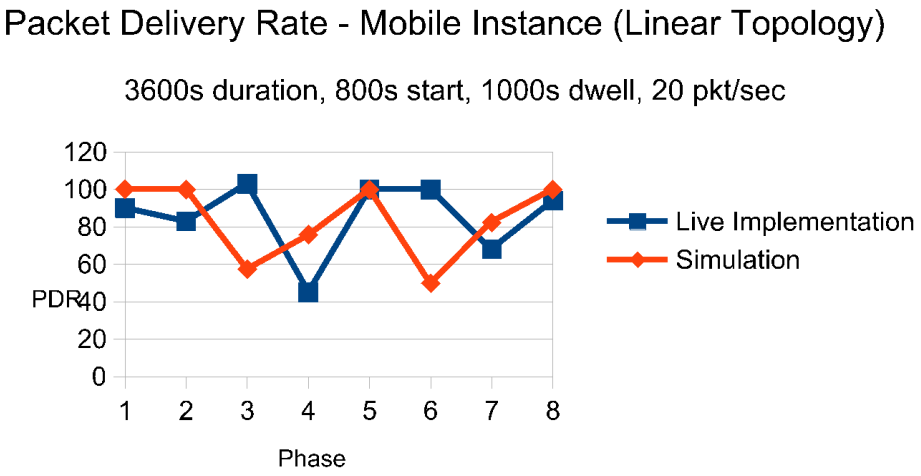

Figure 3: Packet Delivery Rates for the Live and Simulated Mobile Instances

From a practical perspective we were interested in how long the swarm would be disrupted. This time is also known as the *time to convergence.* We also compared our actual results to the widely used Cooja simulator, part of Contiki. Convergence times were measured in the three phases (1, 4, and 7) wherein the robot was motionless and awaiting reform. Figure 2 shows the differences in times to converge for the mobile sink RPL Instance during these three phases for both the live implementation and an average of seven simulated runs using the same scenario in Cooja. As mentioned, the static sink RPL Instance shows negligible interference across both models. Except for phase 7, we have concluded that the Cooja simulated runs show a consistent view of the same test under the same conditions. Packet Delivery Rates (PDR) for the mobile sink were also similar in each phase, as shown in Figure 3. All PDR analyses of the mobile RPL Instance showed a drop during mobility due to parent reconfiguration. Along with other experiments, we

have concluded that a swarm application requiring approximately 3 messages a minute could be reliably supported with disruption periods lasting approximately no more than one minute.

3.2 Simulation Study

We have developed a Cooja-friendly software tool, Tamara, to rapidly generate and evaluate simulated swarm-LLN interactions. Tamara allows us to specify a variety of Flockbot applications and mobility patterns. Using this tool, we evaluated the impact of mobile swarm robots interacting with static robots or LLN nodes in both a linear and a star topology. We are particularly interested in the impact of mobility on convergence time and PDR.

The linear topological model was generated by placing the first sink device at the origin. Each subsequent non-sink node started with an initial placement along a line at 75% of the radio transmission range of the prior placed node. This initial placement location was then adjusted using a random normal distribution provided by the Box-Muller method to add some variance. As Box-Muller produces a normally distributed value with a mean of 0 and unit variance, we multiplied each of the two resulting values by 5% of the transmission range before adding the results to each respective coordinate of the placement location. The process of coordinate adjustment was assessed and repeated if the resulting placement did not lie between 50% and 100% of the transmission range of the prior placed node. The same process was used again for placing the final device, the mobile sink, within transmission range of the final non-sink node on the opposite end of the topology from the starting sink.

Our tests involved a specified dwell period at the target location before returning to the original position.

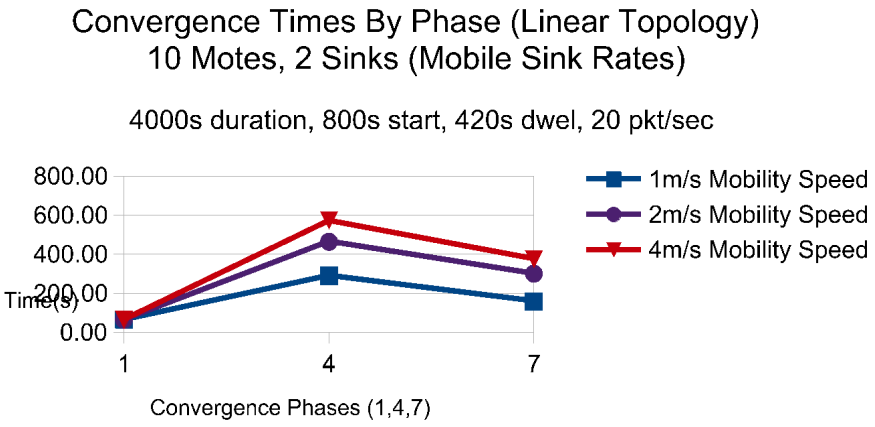

Figure 4: Convergence Times for Various Mobility Speeds

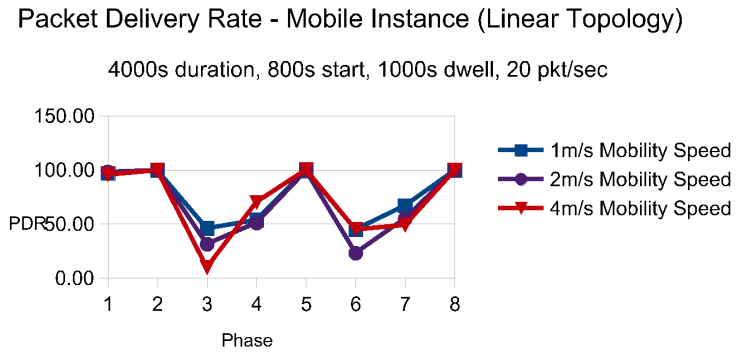

Figure 5: Convergence Times for Various Mobility Speeds

Figure 4 shows the convergence times at each of the key phases for a mobile sink that traverses the topology at a given rate. As expected, the faster the mobile sink traveled, the longer it took to achieve convergence once it arrived at its target destination. Figure 5 shows the impact of node velocity in terms of PDR. As the velocity of the mobile sink increased, it took longer for convergence to occur, which lead to disjoint graphs and undeliverable messages. This effect occurred primarily as a result of the DIO send rate on the mobile sink and was exacerbated by hysteresis in the routing objective function. This lead to nodes accepting the passing mobile sink as a parent and then maintaining that parent selection even after the sink was no longer within range. The results showed that as mobility increased, performance in terms of convergence time and PDR scaled in a roughly linear fashion.

The star topological model was generated by starting with the sink as the origin and used the same method for generating random positions and angles, to place the next device within range of the sink. Once all non-sink nodes were placed within range of the sink, a secondary sink, if selected, was placed in the same manner. This resulted in a star topology for one DAG and a tree topology for the second DAG. Mobility worked in the same manner, selecting a target location within transmission range of a sender node and traveling to that location and dwelling prior to returning.

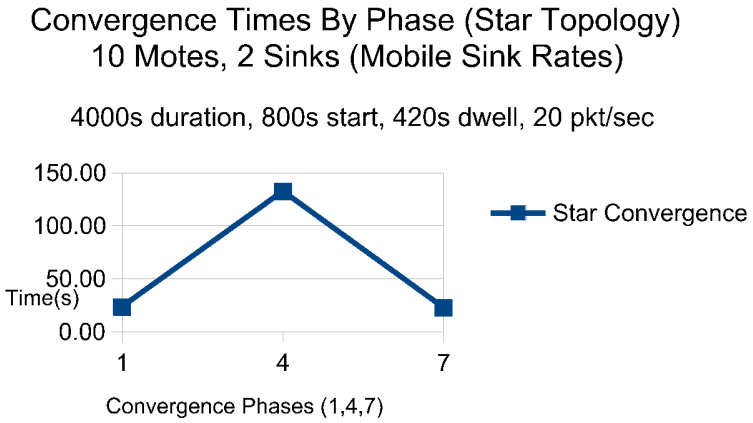

Figure 6: Average Convergence Time for the Star Topology

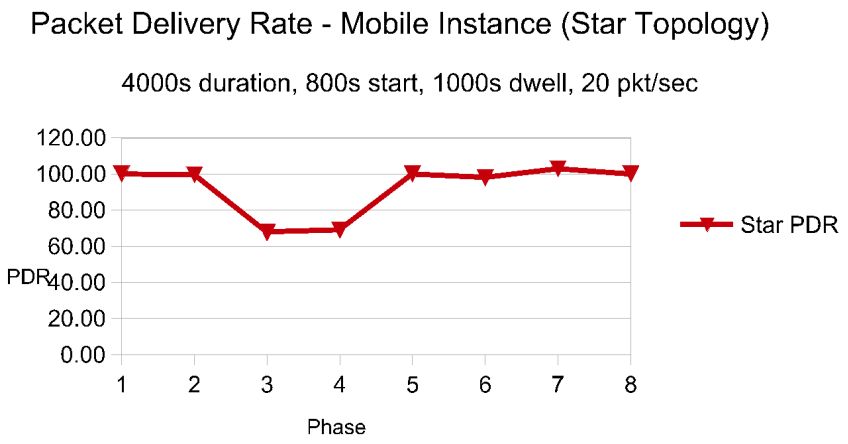

Figure 7: Average PDR for the Star Topology

Figure 6 shows the impact of node velocity in terms of convergence for the star topology. The topology varied in convergence rates based on how the nodes in the star were distributed and how the target location for mobility were selected. As the static sink in this experiment could not belong to the mobile sink Instance, it was entirely possible for the mobile sink Instance to be partitioned in a sparse environment. For these tests, the ten LLN nodes were sufficient to provide good coverage for both Instances. Figure 7 shows the impact of node velocity in terms of PDR. This Figure shows the unique impact of mobility on the star topology. As the mobile sink began to move, it would not follow its own DAG, as was the case in the linear topology. As the mobile sink traveled, it moved to within the range of a node which had formerly been on the opposite side of the DAG, which served to disconnect itself from the entire graph at once, instead of just the portion of the DAG traversed. Figure 7 indicates that PDR dropped more during mobility than was observed in linear topologies.

4 Conclusion

This paper presented the MoRoMi wrapper approach for swarm-LLN communication and coordination. Our results show that swarm to LLN applications can be built using off-the-shelf hardware technologies for scenarios that exhibit both mobile and dwell behaviors. In particular, although network partitions could last up to ten minutes, in all cases the network re-converged. We are currently using MoRoMi to implement a swarm application for mapping and foraging.

Acknowledgements

This work is supported by NSF under grants CNS-1116122, CNS-1205453, RI-0916870, and NRI-1317813. We must also acknowledge the considerable effort of Michael Bowen and David Fleming in updating and redeveloping the Flockbot architecture.

Bibliography

[1] K. Sullivan and S. Luke, “Learning from Demonstration with Swarm Hierarchies,” in *Autonomous Agents and Multiagent Systems (AAMAS)*, Valencia, Spain, 2012.

[2] K. Andrea, D. Fleming and S. Luke, "The FlockBots," George Mason University, Autonomous Robotics Laboratory, 18 February 2016. [Online]. Available at https://cs.gmu.edu/~eclab/projects/robots/flockbots/

[3] T. Winter, P. Thubert, A. Brandt, J. Hui, R. Kelsey, P. Levis, K. Pister, R. Struik, J. Vasseur and R. Alexander, "RPL: IPv6 Routing Protocol for Low-Power and Lossy Networks," RFC 6550, Internet Engineering Task Force, 2012.

[4] B. Coltin and M. Veloso, "Mobile robot task allocation in hybrid wireless sensor networks," in *Intelligent Robots and Systems (IROS)*, Taipei, Taiwan, 2010.

[5] A. Dunkels, "Contiki: The Open Source Operating System for the Internet of Things," 2016. [Online]. Available: http://www.contiki-os.org/. [Accessed 02 May 2016].

[6] T. Watteyne, A. Molinaro, M. Richichi and M. Dohler, "From MANET To IETF ROLL Standardization: a Paradigm Shift in WSN Routing Protocols," *IEEE Communications Surveys and Tutorials,* vol. 13, no. 4, pp. 688-707, April 2011.

[7] K. Heurtefeux, H. Menouar and N. AbuAli, "Experimental evaluation of a Routing Protocol for WSNs: RPL robustness under study," in *IEEE 9th International Conference on Wireless and Mobile Computing, Networking and Communications (WiMob)*, Lyon, France, 2013.