Regression-based Network Monitoring in Swarm Robotic Systems

Josh Rands
Department of Computer Science
Colorado School of Mines
Golden, Colorado
jmrands@mines.edu

Qi Han
Department of Computer Science
Colorado School of Mines
Golden, Colorado
qhan@mines.edu

ABSTRACT

Mobile ad-hoc networks are becoming more common as robotic swarm technology becomes possible. One issue surrounding swarm technology is communication between robots. Communication costs time and energy, and can impact the performance of a swarm. In order to control the network, network state information must be acquired through network monitoring. We propose a novel REgression-based network Monitoring (REM) algorithm where robots in a swarm receive network state data only when necessary for the task at hand. This algorithm will save time and energy in communication by creating predictive models using regressions, minimizing network monitoring overhead.

CCS CONCEPTS

• **Networks → Network monitoring**; *Control path algorithms*; *Network control algorithms*; *Network design and planning algorithms*; Programming interfaces; Network management.

KEYWORDS

network monitoring, swarm robotics, linear regressions, best fit, predictive modeling, mobile ad-hoc networks, distributed networks

ACM Reference Format:
Josh Rands and Qi Han. 2019. Regression-based Network Monitoring in Swarm Robotic Systems. In *Proceedings of the 16th EAI International Conference on Mobile and Ubiquitous Systems: Computing, Networking and Services (MobiQuitous '19).* ACM, New York, NY, USA, 10 pages. https://doi.org/10.1145/1122445.1122456

1 INTRODUCTION

A robotic swarm is a group of locally interacting robots collaborating towards a common goal [5]. These robots can be air vehicles, ground robots, etc. Swarm robotic systems have an overwhelming number of possible applications as they provide many advantages over single robots. Robotic swarms can exploit the sensing capabilities of having a large group; robots in the swarm can specialize their sensors and be built for a specific purpose. Robotic swarms have high situational awareness which allows them to be deployed in highly constrained environments unfit for humans. Potential environments where human deployment is impractical, but robotic swarms can flourish, include: nuclear, chemical, and biological attack detection [5, 21]; battlefield surveillance [5, 21]; search and rescue [5]; and space exploration [5]. In addition, robotic swarms can provide many benefits in environmental monitoring of forests, lakes, etc., and pollution detection [5]. If properly implemented, robotic swarms will be robust to corrupt robots as well as robots leaving or joining the network. Robustness is particularly interesting in tasks such as mine detection. Individual robots in a swarm can sacrifice themselves to locate explosives [21]. Further, robotic swarms are capable of distributing workloads to provide more significant results over large areas. Distributing tasks among robots allows for simultaneous completion of multiple tasks by the swarm.

Effective network monitoring is vital to the success of swarm robotics. The role of network monitoring in swarm robotics is to provide updated state information about robots and networks, i.e., local and pair-wise parameters. Local parameters include robot-specific data such as battery consumption, average data rate, and mobility. Pair-wise parameters include data between two nodes in the network such as received signal strength indicator (RSSI), bandwidth, delay, link quality, and packet loss rate.

Network monitoring can be placed on top of existing infrastructures to provide a service to upper layer protocols or algorithms. In the context of the applications previously discussed, network monitoring provides information to be used for different swarm tasks. When delegating tasks in a swarm, it is vital for a node to be knowledgeable of its peers. One example of how network monitoring information could be used is in coverage planning. It is possible for a coverage planning algorithm to consider the battery level of each node in the network. Another example is Quality of Service (QoS) routing for multimedia data captured by robots in a swarm. Many QoS routing algorithms consider bandwidth and delay constraints between nodes when planning the routing path. Yet another example is point to point routing where the best route is defined as the one that provides the best link quality based on RSSI.

Network monitoring in swarm robotic systems is faced with several interesting challenges, including limited onboard resources, stringent energy constraints, and the dynamic network topology of swarms.

- Network monitoring is a foundation service that supports designated missions. With limited onboard computation and storage resources on each robot, network monitoring must be light weight and cost effective in order to not interfere with the mission.

MobiQuitous '19, November 2019, Houston, Texas, USA
© 2019 Association for Computing Machinery.
ACM ISBN 978-x-xxxx-xxxx-x/YY/MM…$15.00
https://doi.org/10.1145/1122445.1122456

- When a robotic swarm is completing a mission with tight constraints, battery is a major limitation. For a robot, the most energy consuming activity is movement. The second largest source of energy consumption is communication. Energy consumed by computation and sensing is much less than moving and communication. While network monitoring has no control over robot mobility, it can try to reduce communication incurred.
- In mobile robotic swarms, network topology is constantly changing due to the robots' mobility and varying link quality between robots. In addition, robots may join or leave the network at any time. Hence, effective network monitoring solutions must be robust to the dynamic trends of mobile robotic swarms.

To the best of our knowledge, existing distributed swarm robotics methods do not provide an effective solution for network monitoring. Drawbacks to these solutions include unnecessary communication, wasted energy, or even insufficient information. Further details regarding related works are addressed in Section 2.

To overcome the drawbacks of existing approaches, we adopt a distributed hierarchical paradigm. To reduce overhead incurred by network monitoring and to avoid interfering with robot coordination and data communication, we apply the concept of approximate monitoring. Instead of maintaining accurate system state information for each region of the entire environment, we only maintain system status information at a "sufficient" level of accuracy to ensure desired quality. The approximate monitoring is one distinguishing feature of our approach.

2 RELATED WORK

Since robotic swarm systems highly resemble MANETs (Mobile Ad-hoc Networks), we surveyed existing network monitoring solutions developed for MANETs. Network monitoring solutions can be classified based on monitoring architectures, network status data representation, data storage, and monitoring frequency.

Monitoring Architectures. Network monitoring in MANETs use either hierarchical or flat architectures [6]. *Hierarchical approaches* to MANET monitoring are separated by the presence of a central entity.

- In centralized hierarchical approaches [4, 7, 9, 22, 27], nodes are divided into "clusters." Clusters have a node that is designated to be in charge of monitoring that cluster. This node may be referred to as the "clusterhead" [9], a "manager node" [4, 27], or a "domain policy agent" [7]. These approaches are considered centralized because of the presence of a single entity that is above the nodes managing the clusters.
- Decentralized hierarchical approaches [2, 11, 12, 18–20, 25, 26, 28] use distributed algorithms where monitoring is done without a top-level node managing the entire network. These algorithms run on each node in the network without global knowledge of the network. Just like the centralized hierarchical approaches, the distributed hierarchical approaches create clusters that are managed by a manager node. The managing node commonly contains a dominant trait in its cluster [6, 12, 20, 25, 28].

Flat network monitoring architectures [6, 19, 26, 29] distribute network management tasks throughout the network without relying on a clustering topology [6]. Some flat approaches distribute monitoring nodes throughout the network that act as sinks [18, 19, 26]. These nodes gather information for agent nodes as they pass by. Flat architectures are far less common than hierarchical ones. This is because flat architectures have difficulty with scaling. As network size increases, it becomes increasingly difficult to monitor a network using a flat architecture [6].

Monitoring Frequency. The frequency of network monitoring typically falls into one of three categories: event driven, opportunistic, or periodic. Event driven approaches provide network monitoring upon a 'trigger' event [3, 6, 7, 11, 12, 19, 20, 29]. Opportunistic approaches gather data whenever a monitoring agent has the ability to monitor another node in the network [18, 27]. Periodic network monitoring provides monitoring every predetermined time unit [4, 8, 25, 26, 28].

Network Status Data Representation/Model. Our proposed work applies approximation algorithms for different local and pairwise parameters in a network, which is similar to what has been done in the state of the art in creating models for sensornets [14]. One approach uses a time varying multivariate Gaussian with probabilistic confidences to compute sensor models.

Probabilistic models for predicting data in MANETs have been described in [4, 13, 17, 23]. One method is to use a statistical model that uses historical data to predict the next time two nodes will be in contact in a P2P (peer to peer) network [23]. This model calculates the mean estimation for how long it will be until nodes are in contact, as well as a standard deviation to provide error. Another model calculates a spatio-temporal probability that two nodes will be connected, and builds clusters of predicted-to-be well connected nodes [4].

Network Status Data Storage. A common method for storing local and global data in MANETs is through a Management Information Base (MIB). Several different methods use an MIB to keep track of network states [9]. In hierarchical approaches, data is often aggregated by manager nodes [6, 22], the manager locally stores information for all of the nodes in its cluster. In distributed approaches data is stored locally on each node [3, 6, 16, 19, 20, 27, 29].

3 REGRESSION BASED NETWORK MONITORING (REM)

We have designed a novel network monitoring approach to monitor a swarm robotic network in a cost effective manner. Since communication consumes more energy than computation, the REgression-based network Monitoring (REM) approach decreases communication overhead by having nodes predict values of their peers instead of communicating whenever possible, decreasing battery consumption. Saving battery is an absolutely crucial aspect to the success of swarm robotics, and we believe REM can move this evolving field towards a viable and effective network monitoring solution.

3.1 Overview

REM follows a distributed hierarchical approach. We chose a distributed approach because we do not want REM to have a single point of failure from a centralized entity. A distributed architecture is flexible and allows for the immediate deployment of the swarm in any environment. REM follows a hierarchical approach in which nodes in the network are assigned as either a child node or a clusterhead. We chose a cluster-based approach because 1) clustering greatly reduces network monitoring overhead by only sending network monitoring messages to a node's clusterhead and 2) in many situations nodes in a swarm naturally cluster together. For example, in an evacuation, groups of people move as one unit, providing a natural transition to forming clusters.

REM predicts network parameters whenever possible. Models are created using regressions that provide accuracy within a threshold for the task at hand. Child nodes create REM prediction models for local and pair-wise parameters and transmit them to their clusterhead. During runtime, the clusterheads run the REM prediction models of their child nodes. As long as an updated model is not transmitted, the model is assumed to hold true to its estimated accuracy. A flow chart showing how nodes transfer models to their clusterhead is shown in Figure 1.

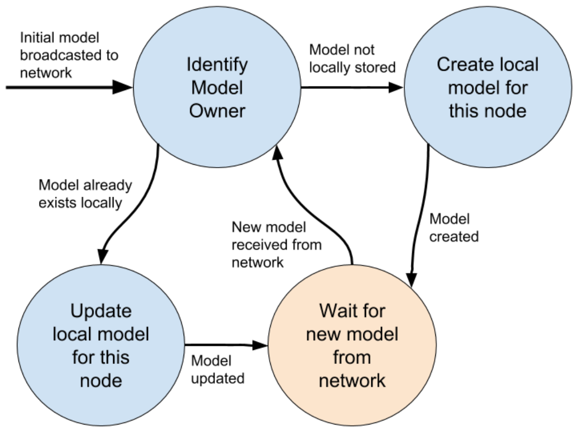

Initial model broadcasted to network
Identify Model Owner
Model not locally stored
Create local model for this node
Model already exists locally
New model received from network
Model created
Update local model for this node
Model updated
Wait for new model from network

Figure 1: Flow chart for clusterheads

Every REM model is built as a function of time. Therefore, at any time t, the clusterhead can calculate the predicted value of different network parameters for its child nodes.

When a node transmits an updated model, it simultaneously begins running that model. While running its own model, the node is comparing its predicted values to the actual measured values. When the measured value falls outside of the error threshold of the predicted value for the model, the node will update the model and transmit the new model to the network.

3.2 Generating REM Models

Historical measured data allows REM to perform best fit regressions. REM can represent large data sets with only three variables: β, μ, and σ. REM generates all predictive models as a function of time. A table showing the different internal parameters for building REM models is shown in Table 1.

Table 1: REM Internal Model Parameters

Variable	Description
Ω	Network wide max error threshold (percentage)
I	Initialize count, number of data points needed for initial model
D_{max}	Maximum difference count
S	Number of standard deviations to fall outside threshold
Δ	Max difference to fall within threshold
W	A list of weighted averages and time stamps
N	Current window size
Q	A list of outliers within W and N
σ_t	A particular model's standard deviation at time t

All prediction models created in REM use best fit regressions. The regression models are performed on I data points. During the network's lifetime, the window size N determines how many points are used to generate new models for models that perform poorly. In this work, we focus on modeling two parameters: remaining battery level as a representative for local data, and received signal strength (RSS) as a representative for pair-wise data. The process for generating a REM prediction model is outlined below. All regressions are performed on a list of weighted averages, W, for a window N. The window size is set at the start of a network's lifetime.

All models take the following form where $P(t)$ is the predicted value at time t and $F(t)$ is some function of time. Equation 1 shows the generic equation for a REM prediction model.

$$P(t) = \mu * F(t) + \beta \tag{1}$$

Creating a REM prediction model follows a three-step process.

Step 1 : Linearize the input data. This is done by taking every data point and replacing the time with $F(t)$.

Step 2 : Perform a linear regression on the linearized data. The linear regression will provide the slope, μ, and the y-intercept, β of the linearized data. The linear regression is performed using the equations outlined below, where x_i represents the i^{th} linearized time value, $\overline{x}$ represents the mean of the linearized times, y represents the measured data points, and n represents the number of data points.

$$S_x = \sum_i x_i, S_y = \sum_i y_i$$
$$S_{xx} = \sum_i (x_i - \overline{x})^2$$
$$S_{xy} = \sum_i (x_i - \overline{x}) * (y_i - \overline{y})$$
$$S_{xx} = \sum_i (y_i - \overline{y})^2$$
$$\mu = (n * S_{xy} - S_x * S_y)/(n * S_{xx} - S_x * S_x)$$
$$\beta = (S_y * S_{xx} - S_x * S_{xy})/(n * S_{xx} - S_x * S_x)$$

Step 3 : Use μ and β to build Equation 1.

3.2.1 *Remaining Battery Level.*

For the remaining battery level prediction model, REM uses a linear regression. The battery level of a node is dependent on three main activities: movement, network communication, and computation. REM assumes that throughout a node's lifetime in a network, these activities will be relatively constant, maintaining a linear drain in battery life. Even if there is variance in the level of battery draining, the REM model will remain valid if the actual value lies within a reasonable distance from the predicted value of the model. If the battery is more dynamic, REM will simply recalculate the model more frequently and still provide valid monitoring of battery level. Equation 2 shows the final form of the REM battery level prediction model.

$$Battery(t) = \mu * t + \beta \tag{2}$$

3.2.2 Received Signal Strength. For the received signal strength prediction model, REM uses a logarithmic regression. A logarithmic regression is performed by linearizing the data, performing a linear regression on the linearized data, and then converting the data back to a logarithm. Equation 3 shows the final form of the REM logarithmic regression model for predicting RSS.

$$RSS(t) = \mu * ln(t) + \beta \tag{3}$$

RSS has been shown to be logarithmically proportional to the distance between the transmitting and receiving nodes [15]. Because the two nodes in this experiment are moving at a constant velocity, RSS is also logarithmically proportional to time, as shown in Figure 2(a).

The next step for generating a logarithmic regression model is to linearize the data. This is done by replacing the time with the natural log of the time (ln(time)). The linearized data is shown in 2(b). Now, a linear regression can be performed on this data, and an RSS REM prediction model can be formed.

3.3 Regression Window

The data in the regression window changes when D_{max} consecutive bad data points have been received and the REM model needs to adapt. There are three different criteria for determining if a REM model no longer fits the incoming data. REM checks all three critera at a predetermined update frequency: 1) The data is an outlier, or falls outside of S standard deviations of the predicted value, 2) The data is outside of the max threshold Δ for the task at hand, and 3) The data is consistently off by similar margin so the model is trending wrong. The trend error uses one standard deviation, or σ as the minimum error for a bad trend remodel to occur. The REM procedure for determining bad data is shown in Algorithm 1.

Algorithm 1 Bad data detection algorithm

```
1: procedure DetectBadDataPoint
2:     deviation ← abs(value - predictedValue)
3:     if deviation > $\Delta$ then
4:         return "Threshold error"
5:     else if deviation > $S * \sigma_t$ then
6:         return "Standard deviation error"
7:     else if deviation > $(S * \sigma_t)/3$ then
8:         if (lastDeviation < 0 AND deviation < 0) OR
9: (lastDeviation > 0 AND deviation > 0) then
10:            return "Bad trend error"
```

Once D_{max} consecutive bad data points are detected, a new model must be created. The node will continue measuring data until it has N data points, build a model, and then broadcast the updated model.

3.4 Error Model

The error of a model is represented by σ. REM calculates the average deviation from the predicted model value for all measured data values within the window of the newly created model. The calculations for σ are shown below, where $P(t_i)$ represents the prediction model value at time t_i, M_i represents the measured value at time t_i, and j represents the data point measured N samples ago.

$$\sigma_{total} = \sum_{i=j}^{j+N} |P(t_i) - M_i|$$
$$\sigma_{model} = \sigma_{total}/N$$

In the case where the average deviation is very low, σ is set to a *minimum deviation* which can be manually set for the application. The default value for *minimum deviation* is 1/5 of the maximum deviation that the task can handle. The goal of setting a minimum deviation is to remove bias in models that calculate unrealistic average deviations. It is possible that N consecutive data values will form a perfect regression that is actually *not* a good fit for future data points.

The error parameter can be used by application layer protocols to determine the accuracy of predicted values by the REM model. Using σ, a range of possible values can be calculated.

3.5 Network Packets and MIB

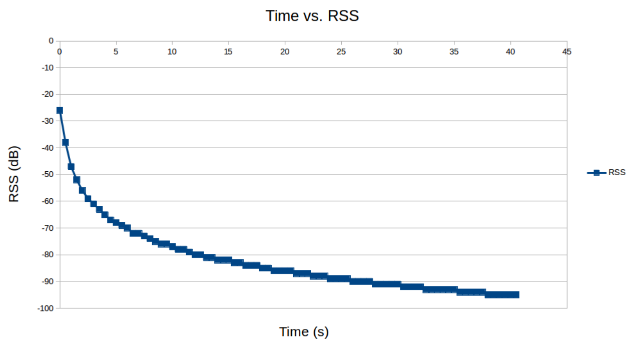
This section describes the structure of REM packets and how clusterheads store REM prediction models in a Management Information Base (MIB). Once REM creates a new model, it must be transmitted to the node's clusterhead. The structure of a REM prediction model packet is shown below.

0 1 2 3 4 5 6 7	8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23	24 25 26 27 28 29 30 31
Node ID	Model Time To Live (s)	Model Type
Mu (float)		
Beta (float)		
Sigma (float)		

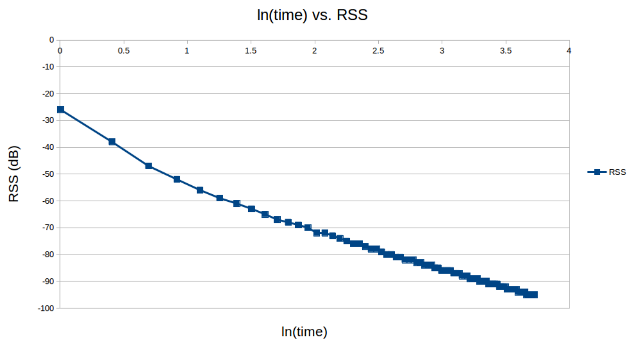
The first byte of the packet is the ID of the parent, or transmitting node. The next 2 bytes are the time to live of the model in seconds. By allowing 16 bits, the model can have a maximum time to live of 2^{16} - 1 seconds, which is approximately 18 hours. The next byte represents the type of model. The next 12 bytes represent the model parameters μ, β, and σ, which are all interpreted as 4 byte floats.

Upon receiving a packet from a child node, the clusterhead stores the REM prediction model in its MIB. The MIB stores one model of each type for each of its child nodes. For example, if node 1 has child nodes 2, 3, and 4, node 1 will store battery models and RSS models for nodes 2, 3, and 4 in its MIB. A child node will transmit a REM RSS prediction model for every node it is connected to.

The MIB is used to provide network monitoring information upon request to other applications in the network. When providing network monitoring information, the clusterhead simply finds the model for the requested node, and calculates the updated value as a function of time using μ and β.



(a) Relationship between received signal strength and time



(b) Linearized data from Figure 2(a)

Figure 2: Process of linearizing measured RSS data

3.6 Model Creation Complexity

Assume a swarm size of S. Each node in the network is creating REM prediction models for all of its neighbors. At any given time, each node can have at most S prediction models. $S-1$ RSS prediction models if the node is connected to every other node in the network, and 1 REM model for local battery drain.

Now, consider that the time complexity to form a prediction model takes time C. The time complexity of REM on a given node in the swarm is $O(S * C)$. C is related to the window size of REM, N. The window size is the number of points that are used to form the regression model. Recall the three step process to creating a prediction model using REM. Step 1 is to linearize the input data, and step two is to run a linear regression. Linearizing the data and finding the mean values $\overline{x}$ and $\overline{y}$ can be done in $O(N)$ time. Once these values have been calculated, performing the linear regression also takes linear time. Finally, calculating the error of the newly formed model can be done in linear time. Therefore, the complexity of REM for a given node is $O(S * N)$.

4 PERFORMANCE EVALUATION

To evaluate our approach, we first validate the REM prediction model using both realistic and synthetic data sets, then identify the best parameters to use for REM, and finally compare REM against a brute-force approach in supporting application level tasks.

4.1 Validation of REM Prediction Models

Two preliminary studies were done to test REM's ability to predict received signal strength between nodes. We tested two scenarios: an infrastructure based network and a peer to peer based network. While we have designed REM with the intention of being used in distributed peer to peer networks, it is important to test its prediction capability in a network with a centralized infrastructure.

The infrastructure based study shows REM's ability to predict the received signal strength of a single mobile node. In addition, this study was used to show that while received signal strength is known to be logarithmically proportional to distance, REM can build prediction models that are logarithmically proportional to time to predict future RSS. In this study, the node is moving in an outdoor environment and communicating with a central access point (AP). This study also showed REM's capability to deal with imprecise, real world data. We compared REM's prediction capability to the predicted RSS of Friis Propagation Loss algorithm [15].

The peer to peer study showed REM's abilty to predict the received signal strength between two or more mobile nodes. This study was a proof of concept to prove that two nodes moving in different directions still have predictable signal strength.

4.1.1 *Infrastructure Based RSS Prediction.*

This infrastructure based study shows REM's ability to predict RSS between a node and a static AP. The study justified REM's performance by comparing the REM prediction model to a well known received signal strength equation: the Friis Propagation Loss algorithm [15]. The Friis Propagation Loss algorithm uses the distance between two nodes to predict the signal strength. The REM model is able to predict the signal strength without knowing the distance between the nodes. This is essential for REM because we want REM to be robust to a variety of environments that may not have GPS capabilities such as underground, space, etc.

This study was done using real world data collected from a mobile robot [24]. The data used was obtained from Crawdad. Using real world data introduced error, thus making the REM prediction model not perfectly accurate. The data set had radio signal strength and location data from a mobile robot in a semi-outdoor environment.

In order to use the data with REM, a modification had to be made. Any data rows where the robot was immobile (velocity = 0 m/s) were removed. REM used time stamps to predict future RSS values. Figure 3 shows the received signal strength predictions from REM and the Friis model.

On average the Friis propagation loss RSS prediction was off by 1.96642 dB and REM was off by 2.25084 dB. When the REM model becomes outdated, or the robot changes direction, there are D_{max} consecutive bad data points before the model adapts. These bad data points significantly contribute to the average error of REM. Without these values, the REM model is much more accurate. This study showed that REM is capable of predicting the received signal

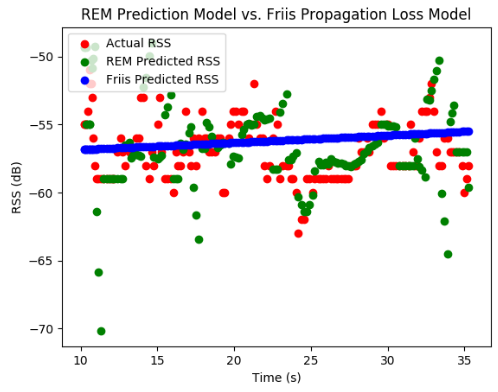


Figure 3: Predicted received signal strength of REM and Friis

strength of a node moving with respect to a centralized infrastructure. This study also showed that while RSS is known to be logarithmically proportional to distance, REM is capable of calculating the RSS between two nodes without knowledge of distance. Because the node are mobile, and distance is a time varying parameter, the distance can be replaced with time. Another important aspect of this study is the use of real world data. Even though the data has a significant amount of error because of natural phenomena, REM is capable of predicting the RSS.

4.1.2 Peer to Peer RSS Prediction. A second study was done to test REM's prediction capability in a peer to peer network. The simulation was done using Network Simulator 3 (NS3).

We created an evacuation scenario as shown in Figure 4. Twelve nodes were placed in the environment in different rooms. This setup allowed some nodes to move in the same direction and some nodes to move in completely different directions. We were able to analyze REM's prediction ability in both scenarios.

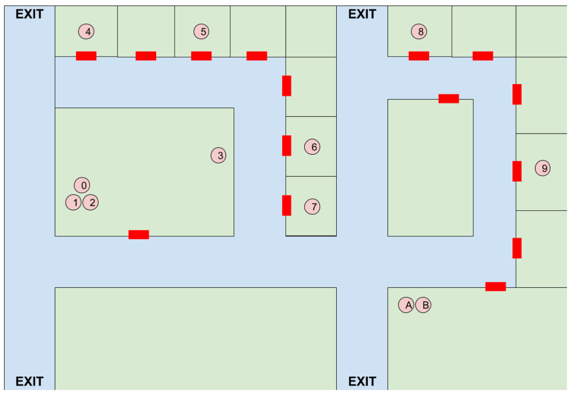


Figure 4: Floor schematics and node locations for the indoor evacuation simulation

The simulation starts when a fire alarm goes off. At this time, every node immediately moves towards the exit of its room. Once in the hallway, the nodes move to the closest EXIT. The average error of REM was 4.2779 dB. Further analysis showed that error was greatly increased because of specific outlier points. Outliers occur when change direction when they are already close together. This movement causes a drastic increase in RSS. REM requires multiple samples to adapt to changes in movements, so for this brief period of time there is a large amount of error. This phenomena can be seen between Node 0 and Node 2 in Figure 5(a) and Node A and Node B in Figure 5(b). When these nodes turned, one node was able to take an 'inside' turn moving the nodes closer together.

In cases where nodes never get close to each other, REM performs very well. This can be seen between Node 0 and Node 7 in Figure 5(c).

This preliminary study showed REM is not only capable of predicting the RSS of a mobile node from a static base station, but it can dynamically predict the signal strength between two mobile nodes with similar accuracy.

4.2 Simulation Scenario

An environmental monitoring scenario was created using Network Simulator 3 (NS3). We were able to modify NS3 source code to provide network monitoring (REM), clustering, and a custom routing algorithm to the drone swarm.

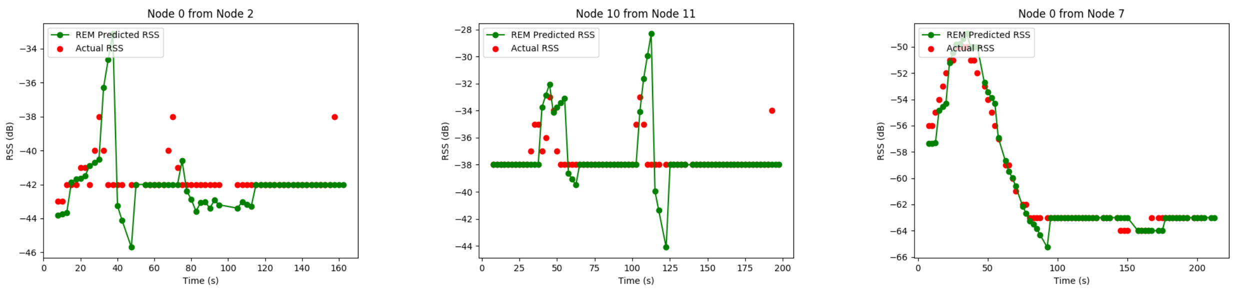
Because REM is providing a service to upper layer protocols, it was created on the application layer. Other layers in the network were created using existing NS3 frameworks. The physical layer was configured to 802.11b for this work. The ad-hoc Wi-Fi MAC layer was used for this simulation. We modified an existing battery model to allow battery drain for network communication and node movement. Drones followed the Random Waypoint Mobility Model which made drones move directly to a random location in the simulation space at a constant velocity.

The simulation scenario consists of a swarm of drones moving around a space with dimensions DxD. There is be a static central server that is collecting environmental monitoring information from drones in the swarm. The server is located at (0, D/2) in the simulation space. The goal of the swarm is to move around the space, collect environmental monitoring data, and then route that information back to the central server for storage.

Nodes in the network were assigned different velocities ranging between 2 m/s and 15 m/s. This is because the velocity range for a standard commercial drone while taking video is 1.5 m/s to 10.5 m/s [1]. In the future, we would like REM to provide network monitoring to allow QoS routing of multimedia data. The max speed of the Parrot Bepop 2 drone is 37.28 mph, which is just over 16 m/s, providing the max velocity cap for these simulations.

We started with **internal** experiments which identified the optimal values for internal REM parameters. Internal experiments tried different combinations of internal parameters within REM, and compared the percent error of REM's prediction compared to the actual measured value.

Once the internal experiments were complete, the most successful REM configuration was compared to our baseline approach in the **external** experiments. The external experiments included clustering and routing. Two metrics were compared between REM and our baseline approach for a variety of different simulation setups: 1) network monitoring overhead and 2) packets delivered to the



(a) Node 0 and Node 2 (b) Node A and Node B (c) Node 0 and Node 7

Figure 5: Predicted vs actual RSS between nodes in building evacuation

central server. The drone swarms success hinges on the number of environmental monitoring packets delivered to the central server.

To enable a complete simulation, the network needed a clustering and routing scheme. Any clustering algorithm could be used for REM. In order to see the impact that REM has on swarm performance, a clustering scheme that relies on connectivity between nodes was used. The clustering scheme used is a Weakly Connected Dominating Set (WCDS) [10]. This clustering scheme assigns clusterheads that create a dominating set across the network with *gateway* nodes between clusterheads. The goal of these simulations is to see the impact of REM on swarm task performance. To do this, clusterheads looked at their children to find a *gateway* node that has the shortest path to the server. For the child nodes, the nextNode is always the clusterhead unless the node is the gateway node, then it is the next node in the path to the server, most often a neighboring clusterhead. When a node comes in contact with the server, the connection is propagated up through the network and shortest paths are updated accordingly.

4.3 Internal Results

First, we evaluated parameters within REM. We performed optimization experiments on three different parameters: 1) the initialize count, I; 2) the max difference count, D_{max}; and 3) the number of standard deviations for a model to become invalid, S. All of these experiments were conducted using the same fixed parameters outlined in Table 2.

Table 2: NS3 simulations environmental variable assignments

NS3 Variable	Value
Number of nodes	20
Simulation dimensions	1000m x 1000m
Node max speed	10 m/s
Node min speed	5 m/s
Simulation Time	300 secs

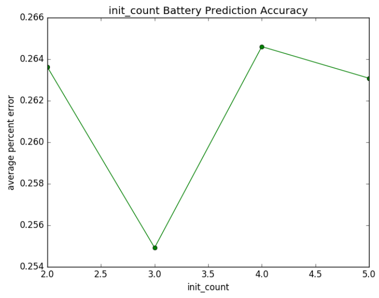
Optimal Initialize Count. The first optimized parameter was the initialize count. We also used this value for the window size when new models are created. These simulations were done with $D_{max} = 3$ and $S = 3$. Four different values for the initialize count were tested, $I = 2$, 3, 4, and 5. To analyze the performance of each value, we calculated the average percent error for every point predicted by REM. The average percent error for the four different initialize count values for predicting battery level and received signal strength can be seen in Figure 6(a) and Figure 7(a).

The results show that the initialize count has a very small impact on the performance of REM. This could be because NS3 provides perfectly consistent RSS data. Therefore, the samples measured always lie on the line of best fit. In practice, it is much more likely for there to be bad samples that do not accurately represent RSS. In this case, a higher initialize count would be needed to account for the variance in the data. However, for this simulation, an initialize count of three seems to yield the most reliable results. This is surprising because one would expect the more samples measured, the lower the error. The reason this is not the case is that the longer REM waits to build the model, the closer the model gets to having to change, resulting in a decrease in the duration of successful predictions.

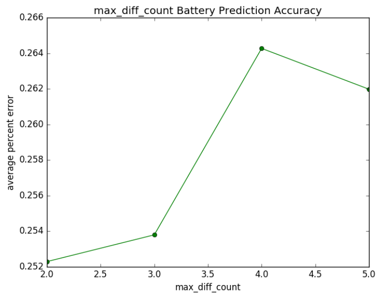
Optimal Max Difference Count. The second optimized parameter was the maximum difference count. These simulations were done with $I = 3$ and $S = 3$. Like the initialize count, four different simulations for the maximum difference count were performed with $D_{max} = 2$, 3, 4, and 5. The results from these simulations can be seen in Figure 6(b) and Figure 7(b).

The optimal max difference count experiments yielded interesting results. As expected, the lower the value of D_{max}, the more accurate the model. This is because REM will recalculate more often, increasing the likelihood of having an appropriate model for the current trend of samples. However, this was not always the case for both experiments. This could be for the same reason as the initialize count.

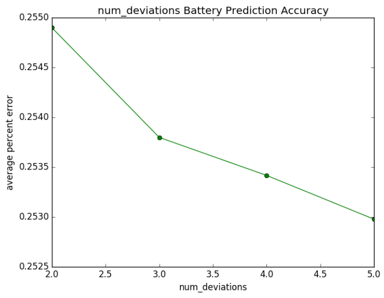
Optimal Number of Deviations. The third internal optimized REM parameter is the number of deviations. These simulations were done with $I = 3$ and $D_{max} = 3$. The results from these simulations can be seen below in Figure 6(c) and Figure 7(c). We hypothesized that a higher number of deviations would result in decreased error. However, the difference between increased number of deviations was less than expected. Figure 7(c) shows that increasing from three deviations to five hardly changes the average percent error of the REM prediction model. Again, this could be because of the nature



(a) Different initialize counts

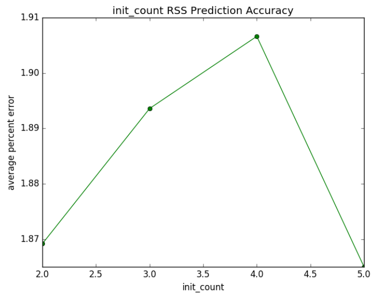


(b) Different maximum difference counts

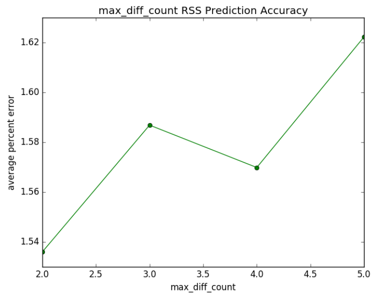


(c) Different number of deviations

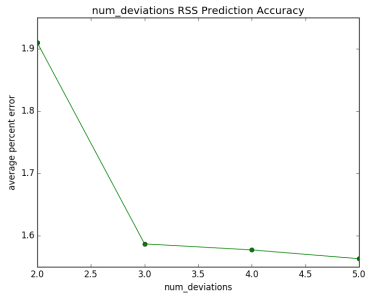
Figure 6: Battery prediction accuracy for different internal parameters



(a) Different initialize counts



(b) Different maximum difference counts



(c) Different number of deviations

Figure 7: Network RSS prediction accuracy for different internal parameters

of a simulation in that it provides perfect data. By increasing the number of required σ counts to recalculate a prediction model, REM increases the chance of creating accurate prediction models.

4.4 External Results

To compare the effectiveness of REM, we developed a baseline "brute force" network monitoring approach with the intention of *not* approximating. The brute force algorithm follows a decentralized hierarchical architecture. This approach uses the same clustering and routing scheme as REM. Nodes enter a broadcasting state every α seconds. In the broadcasting state, nodes will broadcast their locally measured network monitoring values for RSS and battery level. The node's clusterhead will receive this information and store it in an MIB.

We implemented the brute force algorithm in NS3. We compared the success of delivering packets to a centralized server between REM and three different update frequencies for the brute force approach, α = 4, 8, and 12 seconds.

The external simulations compared REM to the brute force approach. For these simulations, we compared the different approaches using average overhead per node and average packets received by the server per node. The hypothesized outcome of these experiments was REM providing a large decrease in overhead, but at the sacrifice of decreased performance. This is because by predicting the state of the network, REM introduces uncertainty and error. The brute force approach also has error, but provides more frequent updates of the network's state. Because of the more frequent updates and providing actual data about the network instead of predictions, we hypothesized the brute force approach will be able to deliver more packets to the central server than REM.

Impact of Node Count. The first parameter that was simulated was node count. We simulated environments of 10, 20, 30, 40, and 50 nodes. All of the environmental parameters except the simulation time were the same for these five experiments. The simulations times for 10, 20, 30, 40, and 50 nodes were 180, 180, 180, 120, and 90 seconds respectively. The simulation space was 1000 m x 1000m and nodes moved at a constant velocity of 7 m/s.

As hypothesized, REM provides much less overhead than the brute force approaches. Figure 8(a) shows the average overhead per node for the different simulations of node counts. REM has significantly less overhead than that of the brute force approaches for all node count simulations. Figure 8(b) shows the average overhead in bytes/node for each of the four simulated approaches. REM has approximately half of the overhead of the brute force approach with the least overhead, where the update frequency was every 12 seconds. This means that the REM prediction models typically lasted longer than 12 seconds. These results are encouraging in

REM's ability to predict the state of a network using regressions on measured RSS data.

Number of packets received by the central server was used to show the performance of these four approaches. As shown in Figure 8(c) and Figure 8(d), all four approaches yield fairly similar results. However, REM and the brute force with $\alpha = 4$ had the highest performances with the most packets delivered.

Impact of Node Density. The next parameter that was simulated was density. For these experiments, the node size was kept the same at 20 nodes, and the size of the space was changed, resulting in a change of density. We simulated environments of 500x500, 750x750, 1000x1000, and 1250x1250 meters. All simulations were 180 seconds long with the nodes moving at 7 m/s. As hypothesized, REM has much less network monitoring overhead than the brute force approaches. These results can be seen in Figure 9(a) and Figure 9(b).

Similar to the node count, the performance of all four approaches was relatively similar. REM had slightly worse performance than the brute force approach with $\alpha = 4$. The results of these simulations can be seen in Figure 9(c) and Figure 9(d).

Impact of Node Velocity. The final parameter that was tested was different velocities. For these simulations, the node count was 20 nodes and the nodes moved in a 1000m x 1000m space. Constant velocities of 2 m/s, 4 m/s, 7 m/s, 10 m/s, and 15 m/s were tested. The velocity range for a Parrot Bepop 2 drone while taking video is approximately 1.5 m/s to 10.5 m/s [1]. The max speed of this drone was the motivation for performing the simulation at 15 m/s.

As hypothesized, REM had less overhead than the brute force approaches. However, it should be noted in Figure 10(a) that when moving at the top speed of 15 m/s, REM's overhead increases and comes closer to the brute force approach with $\alpha = 12$. This is because as velocity increases, the lifetime of REM prediction models decreases. As nodes move faster, they move in and out of range of neighbors more frequently. This results in REM recalculating models, increasing network monitoring overhead. Despite this, Figure 10(b) shows that REM still has significantly less overhead than the three brute force approaches.

Similar to the first two parameters, the performance of all four approaches was relatively similar, with REM providing the most packets delivered to the server. The results of these experiments can be seen in Figure 10(c) and Figure 10(d).

Performance Summary. Overall, REM performed as expected. By predicting the network state with regression-based approximation models, REM decreased network monitoring overhead. REM provided little or no loss in performance when delivering packets to the central server.

5 CONCLUSIONS

This study demonstrated a foundation for the development and application of REM, our REgression-based network Monitoring approach. REM demonstrated capability to predict the RSS and battery level of nodes in a swarm of mobile, networked robots. The robots were collaborating towards a common goal, and REM's imprecise state monitoring provided little hindrance to the performance of the swarm.

Effective network monitoring is vital to the success of swarm robotics, and REM seeks to provide this utility. REM's approximate style network monitoring algorithm reduces overhead, saving energy, by predicting network states whenever possible. We plan on continuing to develop REM and applying the framework to a variety of swarm robotic tasks in highly constrained environments. We hope that REM will help to ensure the successful performance of swarm tasks.

ACKNOWLEDGMENTS

This work is supported in part by NASA SmallSat Technology Partnership (STP) program with grant number 80NSSC18M0048.

REFERENCES

[1] 2019. Parrot BEBOP 2. https://www.parrot.com/us/drones/parrot-bebop-2

[2] Jose Alvarez, Stephane Maag, and Fatiha Zaidi. 2017. Monitoring dynamic mobile ad-hoc networks: A fully Distributed Hybrid Architecture. In *Proceedings of the Advanced Information Networking and Applications (AINA)*. IEEE, 407–414.

[3] Mouna Ayari, Zeinab Movahedi, Guy Pujolle, and Farouk Kamoun. 2009. Adma: Autonomous decentralized management architecture for manets: A simple self-configuring case study. In *Proceedings of the International Conference on Wireless Communications and Mobile Computing: Connecting the World Wirelessly*. ACM, 132–137.

[4] Remi Badonnel, R State, and Olivier Festor. 2006. Probabilistic management of ad-hoc networks. In *Proceedings of the Network Operations and Management Symposium (NOMS)*. IEEE, 339–350.

[5] Jan Carlo Barca and Y Ahmet Sekercioglu. 2013. Swarm robotics reviewed. *Robotica* 31, 3 (2013), 345–359.

[6] Nadia Battat, Hamida Seba, and Hamamache Kheddouci. 2014. Monitoring in mobile ad hoc networks: A survey. *Computer Networks* 69 (2014), 82–100.

[7] Ritu Chadha, Yuu-Heng Cheng, Jason Chiang, Gary Levin, Shih-Wei Li, and Alexander Poylisher. 2004. Policy-based mobile ad hoc network management for DRAMA. In *Proceedings of the Military Communications Conference (MILCOM)*, Vol. 3. IEEE, 1317–1323.

[8] Shigang Chen and Klara Nahrstedt. 1998. An overview of quality of service routing for next-generation high-speed networks: problems and solutions. *IEEE network* 12, 6 (1998), 64–79.

[9] Wenli Chen, Nitin Jain, and Suresh Singh. 1999. ANMP: Ad hoc network management protocol. *Journal on selected areas in communications* 17, 8 (1999), 1506–1531.

[10] Yuanzhu Peter Chen and Arthur L Liestman. 2002. Approximating minimum size weakly-connected dominating sets for clustering mobile ad hoc networks. In *Proceedings of the 3rd ACM international symposium on Mobile ad hoc networking & computing*. ACM, 165–172.

[11] Yuu-Heng Cheng, Abhrajit Ghosh, Ritu Chadha, M Levin Gary, Michelle Wolberg, C Jason Chiang, and Gregory Hadynski. 2010. Managing network security policies in tactical MANETs using DRAMA. In *Proceedings of the Military Communications Conference (MILCOM)*. IEEE, 960–964.

[12] C. Chien-Chung Shen, C. Jaikaeo, C. Srisathapornphat, and C. Zhuochuan Huang. 2002. The Guerrilla management architecture for ad hoc networks. In *Proceedings of MILCOM 2002*, Vol. 1. IEEE, USA, 467,472.

[13] Laurent Decreusefond, T Tung Vu, and Philippe Martins. 2013. Modeling energy consumption in cellular networks. In *25th International Teletraffic Conference (ITC)*. IEEE, 1–7.

[14] Amol Deshpande, Carlos Guestrin, Samuel R Madden, Joseph M Hellerstein, and Wei Hong. 2004. Model-driven data acquisition in sensor networks. In *Proceedings of the Thirtieth international conference on Very large data bases*. VLDB Endowment, 588–599.

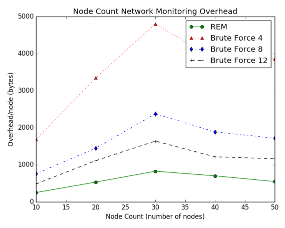
[15] Harald T Friis. 1946. A note on a simple transmission formula. *proc. IRE* 34, 5 (1946), 254–256.

[16] Sana Ghannay, Sonia Mettali Gammar, Farouk Kamoun, and Davor Males. 2004. The monitoring of ad hoc networks based on routing. *IFIP Med-Hoc-NetâĂŹ04* (2004).

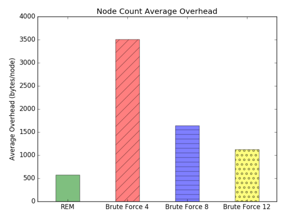
[17] Praveen Gupta and Preeti Saxena. 2010. Energy Consumption in Wireless Ad Hoc Network. In *Proceedings of the 3rd International Conference on Emerging Trends in Engineering and Technology (ICETET)*. IEEE, 831–835.

[18] Hanif Kazemi, George Hadjichristofi, and Luiz A DaSilva. 2008. MMAN-a monitor for mobile ad hoc networks: design, implementation, and experimental evaluation. In *Proceedings of the Third ACM International Workshop on Wireless Network Testbeds, Experimental Evaluation and Characterization*. ACM, 57–64.

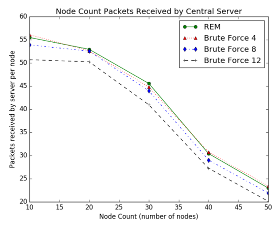
[19] Justin Lipman, Paul Boustead, Joe Chicharo, and John Judge. 2003. Resource aware information collection (raic) in ad hoc networks. In *Proceedings of the 2nd*



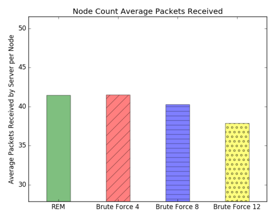
(a) Network overhead for different node counts



(b) Average overhead of each model

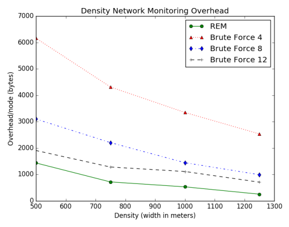


(c) Packets delivered to server for different node counts

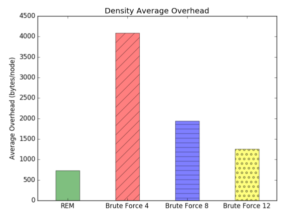


(d) Average packets delivered of each model

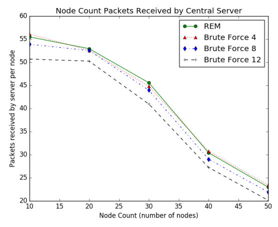
Figure 8: Node count external simulations comparing REM to three brute force models



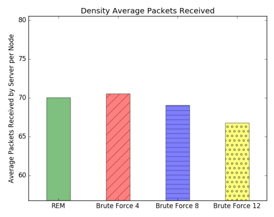
(a) Network overhead for different densities



(b) Average overhead of each model

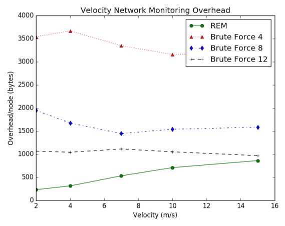


(c) Packets delivered to server for different densities

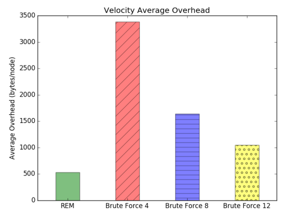


(d) Average packets delivered of each model

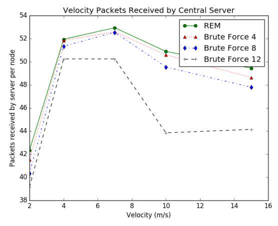
Figure 9: Density external simulations comparing REM to three brute force models



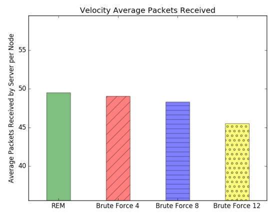
(a) Network overhead for different velocities



(b) Average overhead of each model



(c) Packets delivered to server for different velocities



(d) Average packets delivered of each model

Figure 10: Velocity external simulations comparing REM to three brute force models

Mediterranean Ad Hoc Networking Workshop. 161–168.

[20] Apostolos Malatras, Antonios M Hadjiantonis, and George Pavlou. 2007. Exploiting context-awareness for the autonomic management of mobile ad hoc networks. *Journal of Network and Systems Management* 15, 1 (2007), 29–55.

[21] Iñaki Navarro and Fernando Matía. 2012. An introduction to swarm robotics. *Isrn robotics* 2013 (2012).

[22] Don Ngo, Naveed Hussain, Mahbub Hassan, and Jim Wu. 2003. WANMON: a resource usage monitoring tool for ad hoc wireless networks. In *Proceedings of the International Conference on Local Computer Networks (LCN)*. IEEE, 738–745.

[23] Jéferson C Nobre, Pedro Arthur PR Duarte, Lisandro Z Granville, Liane MR Tarouco, and Fábio Junior Bertinatto. 2014. On using P2P technology to enable opportunistic management in DTNs through statistical estimation. In *Proceedings of the International Conference on Communications (ICC)*. IEEE, 3124–3129.

[24] Ramviyas Parasuraman, Sergio Caccamo, Fredrik Baberg, and Petter Ogren. 2016. CRAWDAD dataset kth/rss (v. 2016-01-05). Downloaded from https://crawdad.org/kth/rss/20160105/outdoor. https://doi.org/10.15783/C7088F traceset: outdoor.

[25] Kaustubh S Phanse and Luiz A Dasilva. 2004. Protocol support for policy-based management of mobile ad hoc networks. In *Proceedings of the Network Operations and Management Symposium (NOMS)*, Vol. 1. IEEE, 3–16.

[26] Krishna N Ramachandran, Elizabeth M Belding-Royer, and KC AImeroth. 2004. DAMON: A distributed architecture for monitoring multi-hop mobile networks. In *Proceedings of the First Annual Communications Society Conference on Sensor and Ad Hoc Communications and Networks (SECON)*. IEEE, 601–609.

[27] Ewerton M Salvador, Daniel F Macedo, José Marcos Nogueira, Virgil Del Duca Almeida, and Lisandro Z Granville. 2016. Hierarchy-based monitoring of vehicular Delay-Tolerant Networks. In *Proceedings of the Consumer Communications & Networking Conference (CCNC)*. IEEE, 447–452.

[28] Dominik Stingl, Christian Gross, Leonhard Nobach, Ralf Steinmetz, and David Hausheer. 2013. BlockTree: Location-aware decentralized monitoring in mobile ad hoc networks. In *Proceedings of the 38th Conference on Local Computer Networks (LCN)*. IEEE, 373–381.

[29] Dominik Stingl, Reimond Retz, Bjorn Richerzhagen, Christian Gross, and Ralf Steinmetz. 2014. Mobi-g: Gossip-based monitoring in manets. In *Proceedings of the Network Operations and Management Symposium (NOMS)*. IEEE, 1–9.