Roadmap Query for Sensor Network Assisted Navigation in Dynamic Environments

Sangeeta Bhattacharya, Nuzhet Atay, Gazihan Alankus,
Chenyang Lu, O. Burchan Bayazit and Gruia-Catalin Roman

Department of Computer Science and Engineering
Washington University in St. Louis

Abstract. Mobile entity navigation in dynamic environments is an essential part of many mission critical applications like search and rescue and fire fighting. The dynamism of the environment necessitates the mobile entity to constantly maintain a high degree of awareness of the changing environment. This criteria makes it difficult to achieve good navigation performance by using just on-board sensors and existing navigation methods and motivates the use of wireless sensor networks (WSNs) to aid navigation. In this paper, we present a novel approach that integrates a roadmap based navigation algorithm with a novel WSN query protocol called Roadmap Query (RQ). RQ enables collection of frequent, up-to-date information about the surrounding environment, thus allowing the mobile entity to make good navigation decisions. Simulation results under realistic fire scenarios show that in highly dynamic environments RQ outperforms existing approaches in both navigation performance and communication cost. We also present a mobile agent based implementation of RQ along with preliminary experimental results, on Mica2 motes.

1 Introduction

Mobile entity navigation is a crucial part of many mission critical applications like fire fighting and search and rescue operations in disaster areas. These scenarios usually involve dynamic environments that make navigation dependent on up-to-date knowledge of the changing environment. Moreover, information about a large region around the mobile entity is required in order to achieve good navigation performance. For example, in the case of a robot navigating a region on fire, the robot would need real-time temperature information about the surrounding areas in order to navigate the region without getting burnt. Also, due to the highly dynamic and unpredictable nature of spreading fire, temperature information of the surrounding areas would be needed frequently for continuous awareness of the neighboring environment. On-board sensors have a limited sensing range and hence cannot provide sufficient information required to make good navigation decisions. Wireless sensor networks (WSNs), on the other hand, present new opportunities to obtain frequent, up-to-date information about a large expanse of the surrounding area. Information obtained from the WSN can be used by the mobile entity to make good navigation decisions, with reduced risks. Moreover, WSNs are easily deployable and are also economically feasible. Once deployed, a WSN can serve several mobile entities and can also be employed to coordinate the movement of multiple mobile entities.

The use of WSNs for navigation in dynamic environments presents important new challenges. Since frequent sensor data updates are required to maintain continuous awareness in dynamic environments, the data collection process can induce a heavy communication workload on the WSN, which usually has limited bandwidth and energy. The resulting network contention and congestion may cause excessive communication delay and loss of sensor data, which may significantly affect the safety and navigation performance of the mobile entity. Therefore, it is important to design efficient query protocols that can collect updated sensor data needed for safe navigation at minimum communication cost.

In this paper, we present a novel roadmap-based approach for navigation in dynamic environments. Our approach consists of two components; a roadmap-based navigation algorithm for the mobile entity and a distributed query protocol called *Roadmap Query (RQ)* for the WSN. The navigation algorithm uses a *roadmap* of the region, which is a virtual graph consisting of possible paths in the region, to search for a safe path to the goal. The path is selected based on roadmap edge weights derived from current sensor data that is collected from the WSN using RQ. RQ achieves communication cost savings by querying nodes only in the vicinity of the mobile entity, called *query area*, and by using a novel sampling strategy that queries only a few selected nodes lying along roadmap edges in the query area. The selective sampling strategy eliminates communication cost resulting from the collection of unnecessary and redundant data, while still enabling RQ to provide sufficient data needed for successful navigation in dynamic environments.

The main contributions of this paper are as follows. (1) We propose a new approach to mobile entity navigation that integrates roadmap based navigation algorithms with distributed query protocols; (2) We present Roadmap Query (RQ), a robust query protocol optimized for navigation in highly dynamic environments; (3) We provide a mobile agent based implementation of a sensor-network assisted navigation system on Mica2 motes; (4) We show through simulations that RQ achieves better navigation performance than existing protocols at only a small fraction of communication cost, in face of realistic fire scenarios and node failures.

2 Related Work

Several methods for robot navigation have been proposed in the past. These methods either assume a priori knowledge of the environment or use on-board sensors to avoid obstacles. A priori knowledge of the environment is not helpful in dynamic environments while on-board sensors have a limited sensing range and hence do not provide information about a sufficiently large region. Recent work in this area suggests integrating WSNs with mobile entities to enable navigation in dynamic environments. The proposed methods fall into two distinct categories.

The first category uses some form of global flooding initiated by the goal, the obstacle or the mobile entity itself. While this approach is effective in relatively static environments, it is unsuitable for dynamic environments since the need to constantly maintain a high degree of awareness of the changing environment (e.g., a spreading fire) would cause frequent flooding of the network. Thus, this approach may suffer from high communication cost and network contention,

which would lead to poor navigation performance. It also wastes energy, thereby decreasing network lifetime. Protocols suggested in [1] and [2] fall into this category. Both protocols construct global navigational fields to guide the robot to the goal. In [1], the goal generates an attractive potiential field that pulls the robot towards the goal, while an obstacle generates a repulsive potential field that pushes the robot away from the obstacle. We will henceforth refer to this method as the Dartmouth Algorithm (DA). Unlike DA, the method in [2] uses value iteration to compute the magnitude of directional vectors that guide the robot to the goal. The approach presented in [3] addresses navigation of mobile sensor nodes, to increase coverage of event locations. In this approach, the goal (an event location) initially floods the network to locate a suitable mobile sensor node. Mobile sensor nodes respond to the flood by sending a response to the goal. The protocol then creates a navigation field around the path taken by the response, to draw the mobile node to the goal. Since the navigational field is only around a path that does not change until the goal changes, this approach cannot efficiently handle dynamic obstacles. Unlike the above approaches, the approach used in [4] assumes that a path already exists in the network, and uses controlled flooding to guide the robot to the start of the path, after which the robot follows the path. This approach is not applicable to dynamic environments where an initially safe path may quickly become unsafe due to changing conditions.

The second category of protocols do not use global flooding but instead use a local query strategy to achieve navigation. Our earlier work, presented in [5], which we will henceforth call Local Query (LQ), falls into this category. In LQ, the path from the start to the goal is built incrementally as the mobile entity traverses the region, by querying *all* nodes in the vicinity of the mobile entity. This approach avoids global flooding by making local decisions. While this method is more efficient than global flooding in a dynamic environment, it still wastes significant amount of energy and bandwidth by unnecessarily collecting information from *all* nodes in the query area. In contrast, RQ uses a selective sampling strategy to collect only necessary information, which is dependent on the environment and changes with it. As a result of this strategy, RQ achieves better navigation performance than LQ at only a small fraction of LQs communication cost (shown in Section 7). This feature makes RQ especially suitable to resource-constrained WSNs. Furthermore, LQ ignores the issue of sensor node failures. In contrast, RQ is designed to handle sensor node failures caused by dynamic obstacles (e.g., being burnt by fire). The robustness of RQ is crucial in such harsh environments where nodes can be easily destroyed. Another new contribution of this work is that unlike LQ, which was implemented in native code, RQ has been implemented using *mobile agents* that can dynamically reprogram nodes in the current query area as the mobile entity moves. An important advantage of our mobile-agent-based implementation is that it enables the adaptive deployment of navigation applications into pre-deployed WSNs with limited resources.

3 Problem Formulation

The navigation problem that we address in this paper is to find a *safe path* for a mobile entity through a sensor field from a start point p_s to a goal point p_g. We

define a *safe path* to be a path that is clear of *dynamic obstacles*, i.e., obstacles whose location or shape changes with time (e.g., car, fire).

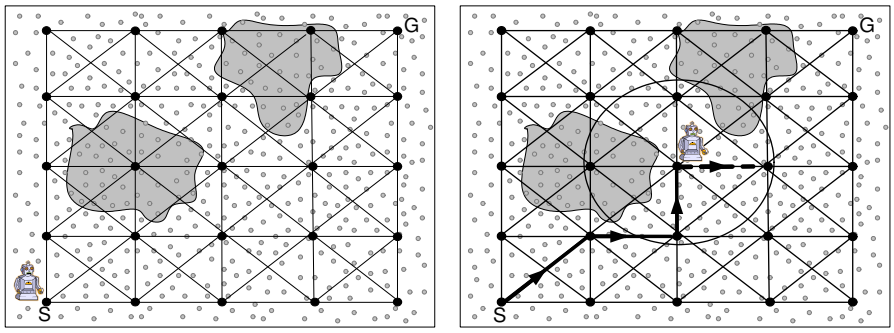
In this paper, we consider fire as the representative example for a dynamic obstacle. Thus, the temperature of the region traversed by the mobile entity is a function of time and is affected by the location and movement of fire. In this case, the problem can be restated as that of finding a safe path for a mobile entity, from start to goal, without the mobile entity getting burnt. The mobile entity is assumed to get burnt if the temperature at its location is higher than a threshold Δ_{burn}. A safe path is now redefined as one where the maximum temperature along the path taken by the mobile entity remains below the threshold Δ_T, while the mobile entity is on the path. Even though our solution is designed assuming fire as the dynamic obstacle, it can be generalized to other types of dynamic environments where safety is defined by changing sensory values (e.g., chemical spills, hazardous gas and air pollution).

We make the following assumptions in the paper: (i) Nodes are location aware. (ii) The mobile entity communicates with the WSN through an on-board gateway device (e.g., PDA) (iii) Nodes have a limited sensing range R_S. R_S is chosen such that if the temperature sensed by a node is below the threshold Δ_T, then the temperature at any point within the sensing range is below the threshold Δ_{burn}. Hence, edges with nodes having temperature above Δ_T are unsafe. The sensing range is thus dependent on the tunable parameter Δ_T. A lower Δ_T results in a longer sensing range.

4 Navigation Algorithm

Our navigation algorithm adapts the roadmap method that is commonly used for navigation in robotics, to make it more suitable for dynamic environments and for integration with WSNs. The roadmap method builds a roadmap of the region and uses it to find a path from the start to the goal. It only considers paths on the roadmap instead of all possible paths in the region and hence, has low computational complexity. Furthermore, it is particularly suitable for WSN assisted navigation, since it reduces the amount of sensor data that must be collected from the WSN by requiring information only along the roadmap edges.

Thus, our navigation algorithm first constructs a roadmap of the region and then incrementally finds safe sub-paths (consisting of roadmap edges) leading to the goal. Sub-paths are selected based on edge weights that are repeatedly updated using temperature information obtained from the WSN, through RQ. The detailed working of our navigation algorithm is as follows. After constructing the roadmap, the mobile entity issues a query to obtain the maximum temperature along the roadmap edges lying within the query area. We assume a circular query area of *query radius* R_q, centered at the current location of the mobile entity $p_e(t)$. After issuing the query, the mobile entity waits for a time T_w to receive the query result which contains the maximum temperatures along the roadmap edges within the query area. At the end of the wait period T_w, the mobile entity computes the edge weights based on the temperature information obtained and finds a safe sub-path to the goal. If the mobile entity finds a safe sub-path, it starts moving along the sub-path. Otherwise, it re-issues the query. This entire process is repeated every time the mobile entity reaches the end of



(a) Roadmap Generation (b) Sub-path Selection

Fig. 1. Working of the Navigation Algorithm. The figures show the grid roadmap. Roadmap vertices and sensor nodes are depicted by black dots and gray dots, respectively. The start and goal are denoted by S and G respectively. The gray shaded regions denote fire. Figure (a) shows the generated roadmap. Figure (b) shows the sub-path selected (dotted arrow) within the query area (shown by the circle). The solid arrows show the path taken by the robot to reach its current location.

a sub-path, until it safely reaches the goal. Note that the navigation algorithm handles the dynamics in the environment by generating the path incrementally, based on fresh information collected from the current query area.

Thus, the roadmap navigation algorithm has three stages, (i) roadmap generation, (ii) roadmap edge weight assignment and (iii) sub-path selection. The roadmap generation stage occurs at the start of the navigation process while the roadmap edge weight assignment and sub-path selection stages, occur repeatedly till the mobile entity reaches the goal safely. These stages are discussed next.

(i) Roadmap generation: We use a grid as the roadmap, as shown in Figure 1(a), where the grid points form the *roadmap vertices* and the edges form the *roadmap edges*. Note that the grid points are virtual points that are placed in space, without considering sensor locations. Traditional roadmap methods (e.g., Probabilistic Roadmap Methods [6]) randomly choose points in space and connect them to construct a roadmap. A benefit of using a grid is that roadmap information can be easily included in a query message without significantly increasing the message size (see Section 5.1). The grid size is a tunable parameter that is a tradeoff between communication cost and navigation performance.

(ii) Roadmap edge weight assignment: The roadmap edge weights are used to find a short, safe sub-path on the roadmap that leads to the goal, as the mobile entity traverses the roadmap. In order to balance path safety and path length, we use an edge weight function that is a weighted function of the normalized maximum edge temperature and the normalized edge length. The maximum edge temperature is provided by the RQ protocol that queries the WSN. The weight of an edge e is thus

$$W_e = \begin{cases} \alpha(\frac{\delta_e}{\Delta_M}) + (1-\alpha)(\frac{l_e}{L}) & \delta_e < \Delta_T \\ \infty & \delta_e \geq \Delta_T \end{cases} \quad (1)$$

where δ_e is the maximum temperature on e based on recent query results, l_e is the length of e, Δ_M is the maximum possible temperature, L is the maximum

edge length among all roadmap edges E and $\alpha \leq 1$ is the weight given to the temperature field. The tunable parameter α determines the tradeoff between safety and path length. δ_e in equation 1 is obtained from the query result and is approximated as $\delta_e = max(\delta_s), s \in S$, where S is the set of nodes that cover edge e and δ_s is the temperature at a sensor $s \in S$. A sensor is said to *cover* an edge if the edge or part of the edge lies within its sensing circle. On the other hand, an edge is said to be covered if certain points on the edge are covered. The points on the edge that need to be covered are determined by the query protocol and will be discussed later. If an edge e is not covered by the nodes that respond to the query, then W_e is pessimistically set to ∞ so as to avoid traversing that edge.

Edge weights are timestamped and expire after a certain interval Δ_{exp}. Δ_{exp} should be chosen carefully, since a large Δ_{exp} will not account for the dynamism of the environment while a small Δ_{exp} may cause the mobile entity to oscillate. Thus, at the end of a query the edges within the query area have edge weights based on up-to-date temperature information obtained from the query result while the edges in the past m query areas have edge weights based on old temperature data, where m depends on Δ_{exp}. All other edges have weights based only on the edge length.

(iii) Sub-path selection: A sub-path consisting of edges lying within the query area is selected by running the Dijkstra's shortest path algorithm [7] on the roadmap. The result of the Dijkstra's algorithm is a path with the least weight from the mobile entity's location to the goal, at that instant. The sub-path consisting of edges within the query area, is extracted from this least-weight path. This stage is illustrated in Figure 1(b).

5 Roadmap Query

In this section, we present the RQ protocol. RQ collects updated temperature information from nodes covering the roadmap edges in a query area. It is issued by the navigation algorithm, every time the mobile entity reaches the end of a sub-path, until the mobile entity reaches the goal. In addition, to improve safety, it is also issued when the temperature at the mobile entity location rises above the threshold Δ_T. The temperature at the mobile entity location is obtained using an on-board sensor.

5.1 Basic Roadmap Query Protocol

RQ minimizes the communication workload on the network by reducing the number of nodes involved in the query process. This is achieved by optimizing the query protocol in accordance with the roadmap-based navigation algorithm. Since the navigation algorithm requires the maximum temperature only along the roadmap edges, the query message, is forwarded only along the roadmap edges lying within the query area. Moreover, due to the high density of sensor nodes, the query message is not forwarded by all nodes along an edge, but only by some selected nodes. These selected nodes form a backbone of nodes along the roadmap edges that fall within the query area and are called *backbone nodes*. RQ requires all backbone nodes to respond to the query. Nodes that hear the

query message but are not on the backbone, respond to the query only if they satisfy a certain criteria and are called *non-backbone nodes*. The backbone and non-backbone nodes form a tree structure with the mobile entity as the root. The formed tree is used to aggregate and deliver the query results to the mobile entity. Thus, RQ reduces communication cost not only by reducing the number of nodes that forward the query message but also by reducing the number of nodes that respond to a query, within a query area.

In order to achieve communication cost reduction, RQ requires the queried nodes to have knowledge of the roadmap and to maintain 2-hop neighborhood information. Since we use a grid as the roadmap, the first requirement is easily met by including the location of the bottom left corner of the grid and the grid square size in the query message. Each queried node uses this information, to calculate the grid points and edges. The second requirement requires all nodes in the network to maintain 2-hop neighborhood information which may introduce some overhead. Neighborhood information is maintained through *hello messages* [8], which contain the ID of the sending node and the IDs of its 1-hop neighbors. Hello messages are broadcasted periodically by each node at an interval called the *hello period*. On receiving a hello message, the receiving node records the sending node as its neighbor and also stores the neighborhood information of the sending node. Each entry in the neighborhood table is associated with a timestamp that corresponds to the time the most recent hello message was received from that neighbor. The timestamp field is used to detect failed neighbors. We have described a simple neighborhood management technique but more sophisticated techniques [9] can also be used. Note that similar neighborhood information is also required by other common services such as routing and power management.

RQ uses two rules to determine which nodes should forward the query message or respond to the query. We call the rule that determines if a node should forward the query message, as the *forwarding rule* and the rule that determines if a node should respond to a query, as the *reply rule*. The forwarding rule identifies backbone nodes while the reply rule identifies non-backbone nodes.

Forwarding Rule: By the forwarding rule, if a node receives a query message that is being propagated along edge $e = \overrightarrow{p_{e1}p_{e2}}$ where p_{e1} and p_{e2} are the endpoints of the edge, and the arrow denotes the direction of query message propagation, then, the node rebroadcasts the message only if it covers edge e and is the closest to p_{e2} among its neighbors that can also hear the same query message. A node knows if a neighbor can hear the same query message by checking its neighborhood table that contains 2-hop neighborhood information. Note that, by this method, only a few nodes along the edge, called backbone nodes, rebroadcast the query message. Thus, some nodes do not rebroadcast the query message, thinking that another node that is closer to the endpoint will rebroadcast the message. These nodes listen for a certain time interval to see if the query message is rebroadcasted. If the query message is not rebroadcasted within this time, these nodes rebroadcast the message. This method takes care of situations where the node selected to rebroadcast the query message does not receive the query message due to collision or other factors.

Reply Rule: The reply rule states that a node should send a query reply, if (i) it is a backbone node, or, (ii) its temperature is above Δ_T and it covers a roadmap edge that falls within the current query area. The first condition draws

```
1. if Query message received
2.     Accept if in current query area.
3.     Set sending node as parent.
4.     Set h_i to hop count in msg plus 1.
5.     Apply forwarding rule to see if msg should be re-broadcasted.
       If yes, re-broadcast msg.
6.     Apply reply rule to see if query result should be sent. If yes,
       calculate time to send result and set timer SendTimer to time-
       out at the right time.
7. else if Query reply received
8.     if result not yet sent then store else discard.
9. else if SendTimer timed out
10.    Send aggregated Query result to parent.
```

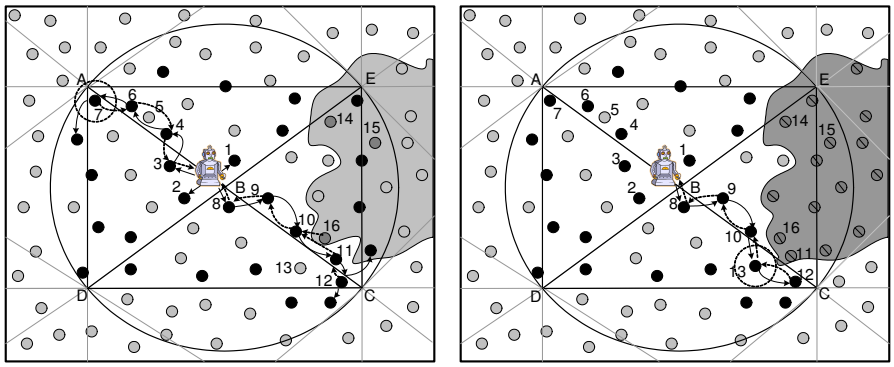
Fig. 2. Roadmap Query (RQ) Algorithm.

query results from the minimum number of nodes that entirely cover all roadmap edges within the query area. The second condition identifies non-backbone nodes and adapts the number of nodes responding to the query, according to the danger level. This condition enforces the safety of the path by drawing query results from nodes if they sense a dynamic obstacle (e.g. fire) near roadmap edges lying within the query area.

Given these two rules, RQ works as follows. On receiving a query message, a node i that lies within the query area, sets the sending node j as its parent, if the link between the nodes is symmetric and sets its hop count h_i to h_j+1 where h_j is the hop count of the sending node and is contained in the query message. The hop count is used to send the query results at a time that facilitates data aggregation. Node i then applies the forwarding rule to determine if it should rebroadcast the message. If it is required to rebroadcast the message, it rebroadcasts the message and then applies the reply rule to determine if it should respond to the query. If it needs to respond to the query, it calculates the time t_r at which the result needs to be sent and sets a timer to timeout at that time. When the timer times out, node i sends its parent an aggregated query result, deduced from its information and the information obtained from its children. If node i and its children are along the same edge, the MAX function is applied to the sensor readings. Otherwise, the results are just merged into one message.

The query reply time t_r is calculated such that it facilitates data aggregation and is set to $t_0 + \frac{h_{max}-h_i}{h_{max}} \times T_w$, where t_0 is the time at which the mobile entity sends the query request and h_{max} is a tunable parameter denoting the maximum possible hop count within a query area. Thus, a node waits for time interval $T_r = t_r - t_c$, where t_c is the time at which the node receives the query message, to receive query replies from its children. The RQ algorithm is shown in Figure 2 and is illustrated in Figure 3(a).

Figure 3(a) illustrates different aspects of the RQ protocol. (i) It shows the backbone (colored black) and non-backbone (colored dark gray) nodes selected by the RQ protocol in response to a query issued by a robot positioned at B. The resulting query area is shown by a solid circle centered at B. The roadmap edges lying within the query area are colored black. (ii) The figure illustrates the query message (solid arrow) propagation along edges BA and BC. The query message is propagated from node 2 to node 3, to node 4 and then to node 6 and finally



(a) (b)

Fig. 3. Working of RQ protocol. The figures show the backbone (colored black) and non backbone nodes (colored dark gray) in a query area (solid black circle) centered at B. Roadmap edges within the query area are colored black. Figure (a) shows the query message (solid arrow) and query reply (dotted arrow) propagation along edges BA and BC. The shaded region represents a region with temperature above the threshold. Figure (b) shows a possible case where RQ fails. The shaded region represents very high temperature at which all nodes in the region (crossed out) have failed. In this case, even though the fire spreads across edge BC, the mobile entity considers BC safe since it hears from enough active nodes with temperatures below Δ_T, that cover BC.

to node 7 along edge BA. The query message propagation along edge BA stops at node 7 since it covers endpoint A, i.e. A lies within node 7's sensing range (shown by dotted circle centered at 7). Node 7 then propagates the message along the adjoining roadmap edges in the query area. Note that the query message is forwarded by node 4 and then by node 6 and not by node 5. This is because of the forwarding rule. When node 5 hears the query message from node 4, it sees that it has a neighbor, node 6, that is also node 4's neighbor (hence, it must have also heard the query message) and that is closer to endpoint A. Thus, by the forwarding rule it does not rebroadcast the query message. (iii) The figure illustrates the outcome of the reply rule when a portion of the query area (shaded region) has temperature above the threshold. By the reply rule, nodes 14, 15 and 16 in this area must reply to the query, since their temperatures are above the threshold and they cover a roadmap edge lying within the query area. These nodes are thus, non-backbone nodes. (iv) The query reply (dotted arrow) propagation along edges BA and BC is also shown. Note how a non-backbone node, node 16, becomes a leaf node, under parent node 10.

5.2 Extension to Handle Node Failures

Robustness to node failures is especially important in dynamic environments since nodes can be destroyed by harsh environments such as fire. The basic RQ protocol cannot handle certain situations arising due to node failures, as shown in Figure 3(b). In the figure, the shaded region depicts a spreading fire, that burns nodes in the region. Due to node failures, edges BE and EC are not covered by working nodes. Hence, the robot does not receive sufficient information about these edges and considers them unsafe. However, edge BC is completely covered

since even though the fire burns node 11, node 13 (which is unaware of the nearby fire) takes its place in forwarding the query message, thus giving the robot the false impression that the edge is safe. If the robot were to choose to traverse edge BC, it would collide with the dynamic obstacle, the fire, and get burnt. This scenario shows the importance of fault-tolerance in dynamic environments. Therefore, we extend RQ to avoid such situations.

In order to make RQ fault-tolerant we include node failure information in the query results. The mobile entity, uses this information to avoid paths with failed nodes, assuming that node failures are due to destruction by fire. Node failure information is obtained, by requiring nodes to send a list of failed neighbors that cover roadmap edges in the current query area, along with their sensor reading. Also, the reply rule is modified slightly such that nodes now send a query reply if (i) they are backbone nodes, (ii) their temperatures are above a threshold and they cover a roadmap edge lying within the query area or (iii) they have failed neighbors that cover roadmap edges lying within the query area.

It can be seen that with these modifications RQ is successful in situations like the one depicted in Figure 3(b). This is because, by the modified reply rule, the robot is informed about the failed nodes 11 and 16 by either node 10, node 12 or node 13. Since node failure is assumed to be due to destruction by fire, the robot infers that the edge BC is not safe and does not traverse that edge. Thus, the modifications make RQ robust to situations where the fire destroys only some nodes along an edge leaving enough working nodes with temperatures below Δ_T to cover the edge, which would give the mobile entity the false impression that the edge is safe.

The modified RQ protocol depends on node failure information, which is easily obtainable. Since each node maintains a neighborhood table and receives periodic hello messages from its neighbors, a node knows if a neighbor has failed, if it hasn't heard from the neighbor in n hello periods. The choice of n has to be made carefully, since a lower value of n will result in more false positives while a higher value of n will result in delayed awareness of danger, thus leading to poor navigation performance. In our simulations, we set $n = 2$.

5.3 Analysis

In this section, we show that RQ successfully gathers the information required by the navigation algorithm within a query area, under the following two conditions.

The first condition is a sensing covered network. In a sensing covered network, every point in the region is covered by at least one sensor. Without this network property, it is impossible to guarantee that a roadmap edge is covered by any sensor at all. A sensing covered network is desirable, as it increases the mobile entity's awareness of the surroundings thus improving its navigation path.

The second condition is the double range property, by which, the communication range R_C of a node is at least twice the sensing range R_S of the node, i.e., $R_C \geq 2R_S$. The double range property guarantees network connectivity in a sensing covered network [10] and hence is a desirable property for such networks. Since the sensing range depends on the temperature threshold Δ_T, we can achieve the double range property by selecting an appropriate Δ_T.

Query message propagation in RQ, starts at a node s that receives the query message from the mobile entity and is closest to the mobile entity's location.

From node s, the query message is forwarded along edges covered by s and then along edges that are connected to them, and so on. Message propagation from one edge to another occurs at nodes that cover the intersection point of two or more edges. Note, that only roadmap edges that *completely* lie within the query area, are considered per query. Since these edges lie completely within the query area, they form a connected subgraph. Given the above, we can prove that "*Given a sensing covered network with $R_C \geq 2R_S$, every node covering a roadmap edge lying completely within a query area receives the query message from the mobile entity, under RQ*". The proof [11] is omitted due to space limitations.

This property of RQ is very useful in environments that do not cause node failures (e.g., chemical spill and air pollution). Environments like fire that cause node failures violate the sensing coverage condition, in which case RQ only provides best effort service. We note that it is extremely difficult to provide any guarantees in the presence of node failures. However, as shown in our simulations, RQ still provides sufficiently good performance in a number of realistic scenarios.

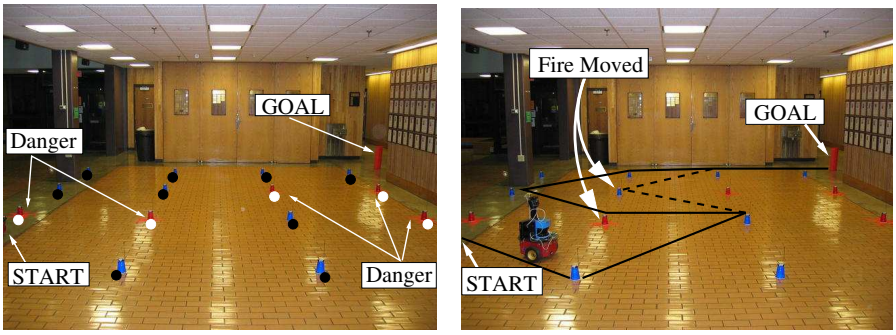
6 Implementation

We implemented the basic RQ protocol on Agilla [12, 13], a mobile agent middleware for the TinyOS [14] platform. A mobile agent based implementation enables RQ to be used in a pre-deployed WSN without requiring the RQ protocol to be pre-installed on the WSN. A WSN running some other application can be quickly re-utilized to run the RQ protocol by just injecting mobile agents containing the protocol into the network. The capability to flexibly reprogram a WSN for a different application is particularly important to WSNs that have limited storage and long operational lifetime [12, 13, 15]. For example, a WSN deployed in a building for temperature monitoring can be quickly re-programmed to run the RQ protocol in case of a fire emergency. The RQ protocol can then be used to guide people safely out of the building.

An Agilla application consists of one or more mobile agents that coordinate with each other, to achieve application-specific behavior. An agent is programmed using a high-level language supported by Agilla. Agilla provides primitives for an agent to move and clone itself from sensor node to sensor node while carrying its code and state, effectively reprogramming the network. New mobile agents can be injected onto a sensor node, thereby allowing new applications to be installed after the network has been deployed. To facilitate inter-agent coordination, Agilla maintains a local tuple space and neighbor list on each sensor node. Multiple agents can communicate and coordinate through local or remote access to tuple spaces. Prior experiences with Agilla have demonstrated that it can provide efficient and reliable services needed by highly dynamic applications such as fire tracking [13].

6.1 RQ using Agents

In the agent based implementation of RQ, the mobile entity injects an *explorer agent* into the network that collects the edge weights and delivers them to the mobile entity. Once injected into the network, the explorer agent clones itself



(a) (b)

Fig. 4. (a) Experimental environment. (b) The robot avoids the initial path (dotted line) and follows a safer path (solid line) when the fire spreads.

on nodes lying along the roadmap edges according to the forwarding rule. The reply rule is applied to determine the agents that need to respond to the query. The agent migration sets up a tree structure along the roadmap edges within the query area, which is used to collect the query result. Per node query results are aggregated such that a list of per-edge-maximum-temperatures is forwarded along the tree branches to the mobile entity through remote tuple space operations. The mobile entity processes the query result and takes appropriate action as explained before.

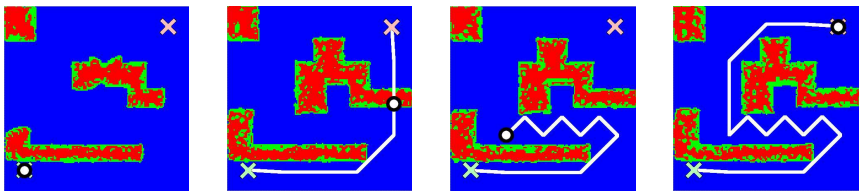
6.2 Experiments

We used a Pioneer-3 DX robot by ActiveMedia [16], as the mobile entity in our experiments. The robot controller carried a mote as a communication interface to a WSN consisting of Mica2 motes. The WSN was arranged in a 4x4 grid, with a grid square length of 2 meters, as shown in figure 4(a). Each node was assigned an (x,y) coordinate based on its position in an euclidean co-ordinate system. In the figure, the node in the lower-left corner was assumed to be the origin of the co-ordinate system with coordinate (0m, 0m). The coordinate of the node in the upper-right corner is therefore (6m, 6m).

The goal of the robot, in the experiments, was to move from (0m, 1m) to (7m, 7m) while avoiding the fire. Experiments were conducted with two types of fire: (a) static fire, and (b) dynamic fire. In the static fire experiments, the temperatures of the motes were fixed throughout the experiment. Fire was simulated by assigning predefined high temperature values (70^oC) to motes located at (0m, 2m), (2m, 2m), (6m, 2m), (4m, 4m), and (6m, 4m) (motes with white dots in Figure 4(a)), and 30^oC to the remaining motes. Dynamic fire was simulated by assigning the same predefined values as in the static fire, but the values were changed during the experiment. More specifically, the temperature of the mote located at (2m, 4m) was increased while the temperature of the mote located at (2m, 2m) was decreased, thus simulating a fire spreading northwards.

Static fire. The path found in this scenario is shown as the dotted line in figure 4(b). As is seen, the robot successfully avoids dangerous places by staying close to motes with normal temperature.

Dynamic fire. The dynamic fire scenario shows the reaction of the robot when the fire changes location. In this case, the robot follows the same path as



(a) Initial state. (b) GQ (c) RQ - intermediate (d) RQ - final

Fig. 5. Path selected by GQ and RQ. (a) Initial state of the environment. The circle depicts the mobile entity; the cross at the bottom left corner marks the starting point; and the cross at the top right corner marks the goal. The blue region represents the safe region and has temperature below 60^oC. The red and green regions represent fire with temperature above 150^oC and above 60^oC, respectively. (b) Mobile entity gets burnt on path selected by GQ. (c) Mobile entity incrementally builds path in RQ. (d) Mobile entity safely reaches the goal on path built by RQ.

the static fire until it reaches (4m, 2m). At this point, the fire at (2m, 2m) moves to (2m, 4m). The robot then successfully finds a new path to avoid the fire. This scenario is illustrated in figure 4(b). The solid line shows the path followed by the robot, while the dotted line represents the initial path.

These experiments demonstrate that a robot can use our agent based implementation of RQ, to successfully find a safe path in the presence of dynamic obstacles. We further evaluate the performance of RQ and compare it with existing approaches through simulations under realistic fire scenarios.

7 Simulation Results

In this section, we present the results obtained from simulations in NS-2. We evaluate and compare RQ with existing protocols, using 9 different realistic fire scenarios, obtained using the NIST Fire Dynamics Simulator (FDS) [17]. In all the scenarios, the fire starts in different locations scattered over the region and then spreads over the region over time. This behavior presents two different environments, (1) which is very dynamic and occurs when the fire is still spreading and most of the region is still not on fire, and (2) which is less dynamic and occurs when a large part of the region is already on fire. We test the performance of the algorithms in both environments, by starting the mobile entity at two different times of 50s and 200s, after the fire starts spreading.

We evaluate the RQ protocol both with and without the extension for handling node failures to observe the difference in performance caused by the extension. We refer to the basic RQ protocol as B-RQ and the RQ protocol with the extension as Robust RQ (R-RQ). We also compare our protocol to other approaches like LQ [5], DA [1] and Global Query (GQ). LQ uses local flooding while DA and GQ use global flooding. LQ and DA were discussed earlier in the related works section (Section 2) and hence are not described here.

In GQ, the mobile entity broadcasts a query message, which is flooded throughout the entire network. On receiving the query message, the nodes respond with their location and temperature. These responses are aggregated and delivered to the mobile entity which uses the data to compute the edge weights of all roadmap edges in accordance with Equation 1 to obtain a complete path

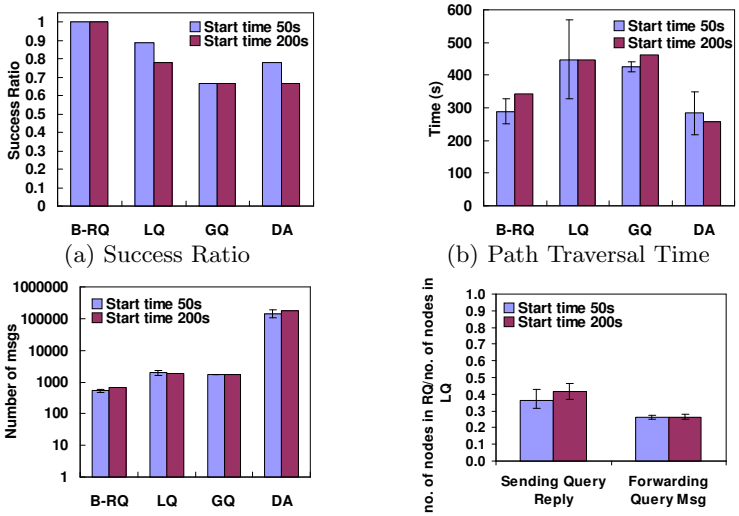
from the start to the goal. Since this method employs global data collection, it has a high communication cost. In addition, it also suffers from a long query latency, which significantly reduces a mobile entity's awareness of the region. This leads to situations where the mobile entity gets burnt while traversing a path that changes from being safe to being unsafe, due to lack of awareness of the changing environment. This situation was observed in a simulation run and is shown in figures 5(a) and 5(b). The white line in Figure 5(b) depicts the safe path that is initially computed by the mobile entity. As the mobile entity traverses the path, a part of the path is engulfed by fire. The mobile entity is unaware of this until it is very close to danger, at which point it stops moving and issues a query to find a safe path. However, due to the significant query latency, the fire spreads to the location of the mobile enitity and burns it, before it finds a safe path. The outcome of using the RQ protocol for the same scenario is shown in figures 5(c) and 5(d). Since the RQ protocol uses local queries with low query latency (due to low communication cost) it computes successive safe sub-paths that successfully lead it to the goal, around the regions on fire.

Each simulation was run with 900 nodes uniformly distributed in a $450m \times 450m$ area. The mobile entity's velocity was set to $3m/s$ and a 5×5 grid was used as the roadmap, with each grid square, measuring $90m \times 90m$. The communication range and bandwidth of the nodes were set to 45m and 40kbps, respectively. The sensing range of the nodes was obtained using the maximum temperature gradient δT at the border of a fire. Thus $R_S = \frac{\Delta_{burn} - \Delta_T}{\delta T}$. δT was found to be $4.5^oC/m$ from the simulation scenarios. Δ_T and Δ_{burn} were set to 60^oC and 150^oC, respectively, assuming a robot as the mobile entity. These settings result in $R_S = 20m$, which satisfies the double range property ($R_C \geq 2R_S$). The query radius (R_q) of B-RQ, R-RQ and LQ was set to 90m in all the simulations, since it was experimentally found to be optimal. Performance under different query radii is omitted due to space constraints but can be found at [11].

As mentioned in Section 4, an edge is considered covered if certain points on the edge are covered. The selection of the points differs with the query protocol. Since LQ and GQ query all nodes within a query area, any number of points (≥ 2) on an edge can be considered in these algorithms. Note that the query area in LQ is a circle (of certain radius) centered at the mobile entity location while the query area in GQ is the entire region. In our simulations of LQ and GQ, we considered coverage of five equidistant points on an edge to indicate edge coverage. However, since fewer nodes respond to queries in RQ, we considered coverage of only the endpoints of an edge to indicate edge coverage in RQ.

The wait time T_w, during which the mobile entity waits for the query results, reflects the query latency in LQ, GQ and RQ. Based on experimental results, it was set to a value that permitted at least 90% query results to be received by the mobile entity, in these protocols. RQ achieves a query latency that is approximately half of that of LQ, due to its low communication cost (Figure 6(c)). Correspondingly, T_w was set to 10s in RQ, 20s in LQ and 250s in GQ.

We use the following metrics to evaluate the performance of the different algorithms. (1) *Success Ratio*, defined as the ratio of the number of scenarios in which the mobile entity safely reaches the goal to the total number of scenarios. This is the most important metric for the application. (2) *Path Traversal Time*, defined as the average time taken by the mobile entity to reach the goal, over scenarios where the mobile entity successfully reaches the goal in all protocols



(a) Success Ratio (b) Path Traversal Time

(c) Communication Cost (d) Per query node participation.

Fig. 6. Performance comparison in the absence of node failures.

being compared. The path traversal time includes the query latency (which depends on the performance of the query protocol) and the time that the mobile entity spends in navigation. (3) *Communication Cost*, defined as the average number of messages sent per scenario, over scenarios where the mobile entity successfully reaches the goal in all protocols being compared.

In the following subsections, in addition to the average results, we also present 90% confidence intervals. Confidence intervals are not provided for the case where the robot start time is 200s and there are no node failures, since there are only two scenarios where the mobile entity safely reaches the goal in all protocols being compared.

7.1 Performance in the Absence of Node Failures

Performance Comparison In this sub-section, we compare the navigation performance and communication cost of GQ, LQ, B-RQ and DA, in the absence of node failures. Since R-RQ is designed specifically to handle node failures, we do not include it in this section.

Success Ratio : Figure 6(a) shows the success ratio of the different protocols at start times 50s and 200s. As seen in the figure, B-RQ performs better than all the other algorithms and enables the mobile entity to safely reach the goal in all tested scenarios. LQ does not perform as well, because unlike B-RQ, it queries all the nodes in a query area and hence incurs a long query latency, which slows down the mobile entity's progress towards the goal. As a result, the mobile entity sometimes gets caught up amidst the spreading fire with no safe path leading to the goal, or gets burnt while waiting for the query results.

The success ratio of GQ is the lowest. This is because GQ does not update the selected path to the goal based on the changing environmental state. At a start time of 50s, the mobile entity usually gets burnt due to lack of awareness of the fire encroaching the chosen path. On the other hand, at a start time of 200s,

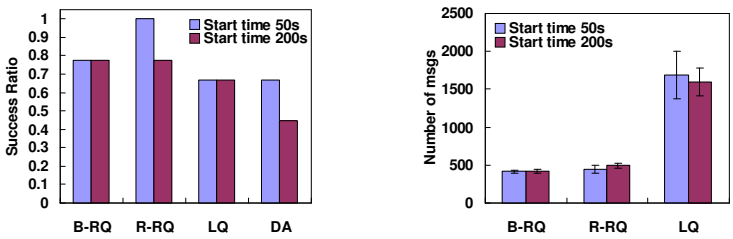
the mobile entity usually fails to obtain a safe path, because, by the time the mobile entity obtains the query results, which can take as long as 250s, most of the region is already engulfed by fire, disconnecting the start from the goal. The success ratio of DA is lower than that of B-RQ and LQ, because of high data loss due to contention caused by the high communication cost of DA (Figure 6(c)).

Path Traversal Time : Figure 6(b) shows the path traversal time obtained by the different protocols. We see that DA achieves the least path traversal time. This is because DA has very low query latency compared to the other algorithms, as it requires the mobile entity to query only nearby nodes. The path traversal time of B-RQ is comparable to that of DA. This is because, B-RQ also has a low query latency, since it queries only a few nodes per query. LQ and GQ, on the other hand, have longer path traversal times, since they query a large number of nodes and hence have long query latencies. Since all protocols achieve similar path lengths (shown in [11]), the difference in the path traversal times is dominated by the difference in query latencies.

Communication Cost : A comparison of the communication cost is presented in Figure 6(c). DA has an extremely high communication cost, since it requires all nodes in the network to maintain the potential fields, resulting in frequent flooding of the entire network caused by the spreading fire. In comparison, B-RQ has the least communication cost, since it queries only a few nodes that lie along the roadmap edges in a query area, per query. The extent by which RQ reduces the number of nodes that participate in a query, in comparison to LQ is shown in Figure 6(d). The figure shows that the number of nodes that forward the query message, per query in RQ, is only about 25% that of LQ and the number of nodes that send a query reply, per query in RQ, is only about 40% that of LQ. Thus, the total number of nodes that participate in a query in RQ is only about 40% that of LQ, on average. This significant reduction in the number of participating nodes per query is the reason behind the dramatic reduction in communication cost achieved by RQ. This saving is found to be 73% for a start time of 50s and 63% for a start time of 200s, from Figure 6(c). As a result of the reduced communication cost, RQ also has a lower query latency in comparison to LQ (almost 50% that of LQ), thus increasing its navigation performance, in terms of success ratio and path traversal time. This implies that greater the dynamism of the environment, the better will RQs performance be, in comparison to LQ. Another positive outcome of reducing the number of nodes participating per query is that when coupled with a power management protocol, it enables more nodes to sleep, thereby increasing network lifetime.

As expected, B-RQ also achieves huge communication cost savings over GQ. In particular, it achieves 70% and 62% lower communication cost for a start time of 50s and 200s, respectively. Note that, the large differences in communication cost between B-RQ, LQ and GQ, are not clearly visible in Figure 6(d), due to the logarithmic scale. The low communication cost of B-RQ is one of its main advantages, which enhances its navigation performance and also potentially increases network lifetime.

Overall, B-RQ achieves a higher success ratio and a significantly lower communication cost than all the other protocols as a result of its efficient forwarding and query reply rules. These results highlight the effectiveness of optimizing the



(a) Success Ratio (b) Communication Cost

Fig. 7. Performance comparison in the presence of node failures.

query protocol in accordance with the navigation algorithm, in order to navigate successfully in dynamic environments.

7.2 Performance in the Presence of Node Failures

Since it is important to design robust protocols that can tolerate node failures caused by harsh environments, we now compare the performance of the algorithms in the presence of node failures. Nodes are assumed to fail at a temperature of 150^oC.

Performance Comparison In this sub-section, we compare the navigation performance and communication cost of LQ, DA, B-RQ and R-RQ. GQ is not considered in these simulations, due to its poor performance in the earlier simulations.

Success Ratio : As shown in Figure 7(a) R-RQ effectively improves the success ratio of B-RQ when the mobile entity starts at 50s, at which time many new nodes start failing, due to the spreading fire. This is because R-RQ informs the mobile entity about failed nodes, thus warning the mobile entity about danger areas. On the other hand, R-RQ does not outperform B-RQ when the mobile entity starts at 200s, since by that time, the environment is relatively stable. In contrast, the performance of LQ and DA are affected significantly by node failures. R-RQ achieves upto 49% improvement in success ratio over LQ and up to 77% improvement in success ratio over DA. This demonstrates that R-RQ is particularly important in dynamic environments.

Communication Cost : Figure 7(b) shows the communication cost incurred by B-RQ, R-RQ and LQ. The commmunication cost of DA is not shown as it is significantly higher than the other protocols. As expected, the communication cost of LQ is much higher than that of B-RQ and R-RQ since it queries all the nodes in a query area. The communication cost of R-RQ is only slightly more than that of B-RQ since it requires nodes to respond if they have failed neighbors even if their temperatures are below the threshold. More specifically, R-RQ achieves 73% and 69% savings in communication cost over LQ, at a start time of 50s and 200s, respectively.

8 Conclusion

In summary, we propose a novel approach that integrates roadmap-based navigation with efficient query protocols for navigation in *dynamic* environments.

We present the Roadmap Query (RQ) protocol that is specially optimized for collecting fresh data needed for navigation in the presence of dynamic obstacles and sensor node failures. We also present a mobile-agent based implementation of our navigation approach on a physical testbed consisting of Mica2 motes and a robot. Our simulation results demonstrate that RQ can significantly improve the success ratio of navigation while introducing minimum communication cost under realistic fire scenarios and node failures. Our results highlight the importance of joint optimization of navigation and WSN query protocols for efficient navigation in dynamic environments.

Acknowledgement

This work is funded in part by the NSF under an ITR grant CCR-0325529 and the ONR under MURI research contract N00014-02-1-0715.

References

1. Li, Q., Rosa, M.D., Rus, D.: Distributed algorithms for guiding navigation across a sensor network. (In: MobiCom'03)
2. Batalin, M.A., Sukhatme, G.S., Hatting, M.: Mobile robot navigation using a sensor network. (In: ICRA'04)
3. Verma, A., Sawant, H., Tan, J.: Selection and navigation of mobile sensor nodes using a sensor network. (In: PerCom'05)
4. Corke, P., Peterson, R., Rus, D.: Coordinating aerial robots and sensor networks for localization and navigation. (In: DARS'04)
5. Alankus, G., Atay, N., Lu, C., Bayazit, B.: Spatiotemporal query strategies for navigation in dynamic sensor network environments. (In: IROS'05)
6. Kavraki, L., Svestka, P., Latombe, J.C., Overmars, M.: Probabilistic roadmaps for path planning in high-dimensional configuration spaces. IEEE Trans. Robot. Automat. **12**(4) (1996) 566–580
7. Cormen, T.H., Leiserson, C.E., Rivest, R.L.: Introduction to algorithms. 6th edn. MIT Press and McGraw-Hill Book Company (1992)
8. Whitehouse, K., Sharp, C., Brewer, E., Culle, D.: Hood: a neighborhood abstraction for sensor networks. (In: MobiSys'04)
9. Woo, A., Tong, T., Culler, D.: Taming the underlying challenges of reliable multhop routing in sensor networks. (In: Sensys'03)
10. Xing, G., Wang, X., Zhang, Y., Lu, C., Pless, R., Gill, C.: Integrated coverage and connectivity configuration in wireless sensor networks. TOSN **1**(1) (2005) 36–72
11. Bhattacharya, S., Atay, N., Alankus, G., Lu, C., Roman, G.C., Bayazit, B.: Roadmap query for sensor network assisted navigation in dynamic environments. In: Technical Report WUCSE-05-41, Department of Computer Science and Engineering, Washington University in St. Louis. (2005)
12. Fok, C.L., Roman, G.C., Lu, C.: Rapid development and flexible deployment of adaptive wireless sensor network applications. (In: ICDCS'05)
13. Fok, C.L., Roman, G.C., Lu, C.: Mobile agent middleware for sensor networks: An application case study. (In: IPSN'05)
14. : (TinyOS community forum) http://www.tinyos.net/.
15. Levis, P., Culler, D.: Mate: A tiny virtual machine for sensor networks. (In: ASPLOS X)
16. : (Activmedia) http://www.activemedia.com.
17. McGrattan, K.: Fire dynamics simulator (version 4) technical reference guide. National Institute of Standards and Technology (2004)