TIME-SPACE MULTICAST FOR INEGRATED SENSOR/BATTLESITE NETWORKS

Hu Zhou and Suresh Singh
Department of ECE
Oregon State University

Abstract— *An important objective of tactical ad hoc networks is to deliver threat information from sensors to shooters efficiently and quickly. The information sent to a particular shooter should contain warnings about threats that are within some* distance *and/or within some* time *of the shooter's current location. In this paper we develop a novel multicast model that distributes this form of threat information in a message efficient manner. We present results from detailed simulations that demonstrate the efficiency of our protocol and discus the scalability of this model to larger networks.*

I. INTRODUCTION

In battle, shooters need to be constantly informed of impending enemy threats and information such as time-to-threat, distance-to-threat and type-of-threat must be presented to them in a timely manner. This type of information is typically gathered by sensors and uploaded into the ad hoc battlesite network(s). It is then the task of the network protocols to then send this information to shooters (in addition to warfighters, etc.) in a timely manner. The multicast model we discuss in this paper is concerned with distributing threat information to shooters based on their individual *time* and *distance* to threats. In other words, a shooter is typically most interested in immediate threats (within, say, a ten minute time horizon) rather than in more distant threats (that may be an hour away). Therefore, the multicast protocol must forward data to the individual shooters based on *their* individual time/distance to different threats in the battlespace. This requirement is unique and distinguishes our *time-space multicast* model from all other multicast models (section II).

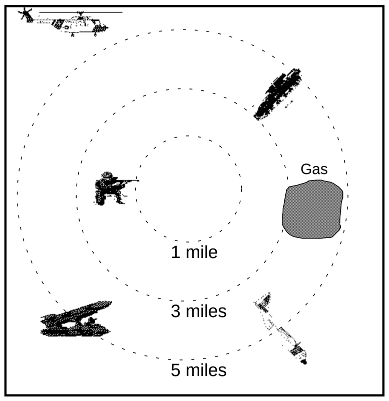
In order to describe the problem of T-S multicast, it is helpful to consider a simple example. Figure 1 illustrates two hypothetical views of the battlespace that could be presented to the shooter. The first view indicates the distance to different types of threats and the second view indicates the time to different threats. Thus, there is a gas cloud three miles away but, due to the wind direction and speed, it is less than one minute from the shooter. Likewise, the helicopter is over five miles away but, because of its speed, it will pose a threat in between one and five minutes. The information most relevant to the shooter can thus be classified as:

- the location of all threats that are within t seconds of the shooter's current position, and
- the location of all threats that are no more than d meters away.

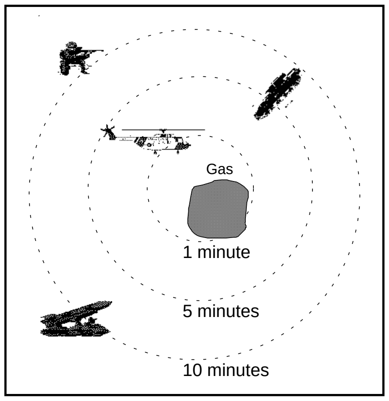
It is noteworthy that shooters will dynamically change the t and d specifications depending on battle conditions and, furthermore, different shooters will typically have different t and d specifications. Thus we see that delivering information from sensors to shooters in a way that satisfies each shooters t and d specifications is a non-trivial problem.

A. Challenges

In order to explain the challenge in building a multicast protocol that satisfies the t and d specifications of different shooters, let us look at some examples. For the sake of clarity, let us initially assume that the t and d specifications for all shooters is the same.



(a) Distance to threat



(b) Time to threat

Fig. 1. Two views of the battlespace presented to the shooter.

Consider Figure 2. Assume that a sensor detects a gas cloud in the area and determines that the wind is blowing from the east. In this case, the sensor needs to multicast this information in the westerly direction only. If the wind changes direction and begins blowing from the south west, however, the sensor will need to multicast its information to receivers located in a north easterly direction. Thus, the *direction* of the threat influences the conduct of the multicast. A second consideration here is *how far* should this information be propagated in the network? In Figure 2, everyone in region A needs to be warned (if we assume $t = 10$ minutes) but region B is not in immediate (i.e., ten minutes) danger so there is no need to extend the multicast to nodes here.

Let us consider another situation , illustrated in Figure 2, where a car is travelling towards the gas cloud at speed. Here it is necessary to ensure that the multicast reaches the car because it will be in danger within 10 minutes. Unfortunately, however, the sensor does not know about the existence of the car and cannot include it in its multicast. Thus, there is a need to extend the multicast from the sensors so that *mobile nodes*, such as the car, also receive the information even though they are initially far away. One method of ensuring that the multicast reaches the car is to extend the multicast to region B. Unfortunately, this represents a high message cost.

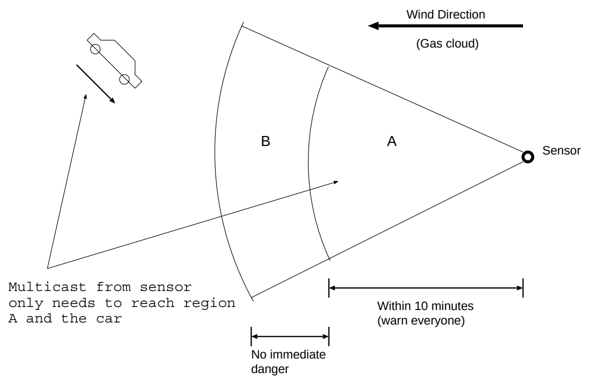

Wind Direction
(Gas cloud)
B
A
Sensor
Multicast from sensor only needs to reach region A and the car
Within 10 minutes (warn everyone)
No immediate danger

Fig. 2. Multicast coverage of information generated by a sensor.

Before we discuss an approach to solving this problem, however, we need to consider one final aspect of the model, *viz.*, the effects of the *enemy weapons* on the conduct of the multicast. An enemy shooter (on foot) may not pose a threat to a tank but will pose a threat to other similarly equipped shooters who come within, say, two hundred meters. On the other hand, an enemy aircraft that is far away and that may not even be on an intercept course will pose a threat to the tank because of the aircraft's ability to fire smart air-to-surface missiles. Thus, when identifying threats that are t seconds or d distance away, it is important to consider the "strike range" as well as the effectiveness of such weapons.

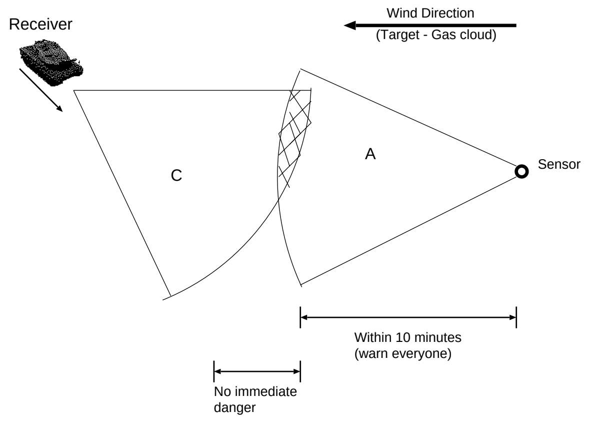


Fig. 3. Multicast data collected by a receiver via receiver-pull.

In summary, a shooter needs to have information about threats that are within time t seconds or within distance d. This information is affected by two factors:

- The *relative velocity* of each enemy w.r.t. the shooter, and,
- The strike range and effectiveness of the enemy weapon.

The information regarding threats is collected by sensors and then needs to be multicast to all those who need it (based on t and d specifications). This problem of multicast is made difficult because:

- The sensors do not know the composition of their multicast groups since group membership is based on the t and d specifications of individual receivers.
- Changes in the troop/threat deployment changes the t and s specifications.

Our approach for developing a solution to the T-S multicast problem discussed above is to use a *sensor-push receiver-pull* approach. Here, sensors push the information out into the network to some distance and receivers then pull relevant information from the network satisfying their time-space mandates. In Figure 3, for instance, the sensor pushes out the information about the gas cloud into region A. The tank sends a request for information *ahead* of it to nodes within distance vt meters (the tank's speed is v meters/sec). In Figure 3 this is represented by region C. All multicast information available to nodes in region C is forwarded to the tank. Thus, if there is any node in the intersection of regions A and C, information about the gas cloud will get forwarded to the tank. We will expand this idea in section III.

B. Overview of the paper

- In section II we describe other multicast protocols for ad hoc networks and describe how our time-space model differs from them.
- Section III describes our multicast protocol in detail.
- We present results of simulations in section IV.
- The work reported in this paper is ongoing and we describe our current research focus in section V.

II. Related Work

Recently several authors have begun developing multicast protocols for ad hoc networks. The primary challenge they have attempted to solve has been to construct and maintain multicast *trees* despite topology changes. Some examples of these protocols include, *AMRoute (Adhoc Multicast Routing)* [1], *ODMRP (On-Demand Multicast Routing Protocol)* [3], *AMRIS (Ad hoc Multicast Routing protocol utilizing Increasing id-numberS)* [4], and, *CAMP (Core-Assisted Mesh Protocol)* [2]. AMRoute and AMRIS build multicast trees to forward data while ODMRP and CAMP use mesh structures to forward multicast data.

Unlike these protocols, the T-S model proposed in this paper is very different because, in our case, the *multicast receivers* of a data stream are determined based on the *data contents* of individual data packets (recall that a receiver only wants to receive data that informs it of impending threats defined in time or space). As the data content from a sender changes, so does the multicast group! This feature of our model makes it unique as well as extremely powerful in the military context where the goal is to maximize message efficiency while ensuring that the *sensor-to-shooter distance* is minimized.

III. Description of the T-S Multicast Protocol

Our multicast protocol is based on the idea of *sensor-push* and *receiver-pull*. Sensors detecting threats send a limited broadcast message into a small region that lies in the path of the threat and individual receivers then pull threat warnings from nodes that lie in the direction of their travel. This push-pull strategy works efficiently because we reduce the number of unnecessary threat warning messages by ensuring that the only nodes that need to receive the warning receive them.

For an efficient implementation of the push-pull approach, we view the battlefield as being divided into geographic regions as shown in Figure 4. These regions are virtual and are only used for improving the message efficiency of our protocol. It is important to note that the regions do not need to be of the same shape or even size. The specific form of regions will be based on the density of nodes within the region, terrain characteristics, and node mobility constraints.

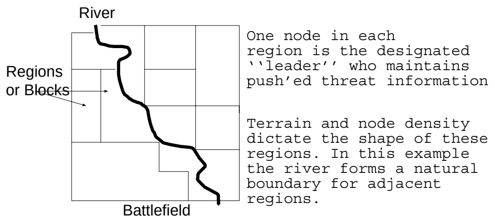

River
Regions or Blocks
One node in each region is the designated ``leader'' who maintains push'ed threat information
Terrain and node density dictate the shape of these regions. In this example the river forms a natural boundary for adjacent regions.
Battlefield

Fig. 4. The battlefield is viewed as being made up of regions.

Within each block one node is chosen to be the "leader". This node maintains a list of all threat warnings received via push messages and is responsible for responding to pull requests as well. We will discuss these two parts of the protocol in sections III-A and III-B. Section III-C describes the leader maintenance protocol.

A. Push Protocol

Warning messages regarding threats are generated by sensors as and when a new threat is detected or the disposition of a known threat changes (e.g., it changes direcion). Thus, when a sensor or a collection of sensors detects the presence of a threat, the sensor generates a limited broadcast message for nodes that will lie in the *projected path of the threat*. Consider Figure 5 where, for the sake of clarity, we assume that the blocks are all identical rectangles. When a tank threat is detected, the sensor(s) that detected the threat need to inform nodes that lie in the path of the tank of a possibly imminent attack by the tank. In the figure we indicate that the extent of this warning push message extends out to blocks that lie within distance $s(\tau_k + \Delta\tau)$. Here s is the speed of the tank, τ_k is a constant and denotes the *time* specification that is used by tank sensors to determine the extent of the push. In our simulations we use a value of $\tau_k = 300$ seconds and the interpretation is that the push informs nodes that are no more than 5 minutes away from encountering the tank threat. The $\Delta\tau$ factor is an error factor (and is also a constant) that ensures that more rather than fewer nodes get informed of the threat.

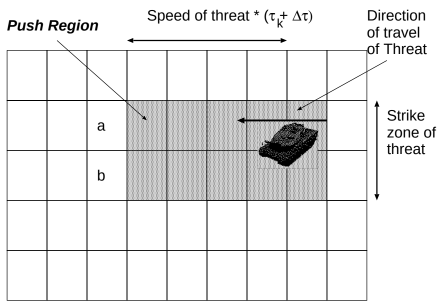

Push Region
Speed of threat * (τ_k + $\Delta\tau$)
Direction of travel of Threat
a
b
Strike zone of threat

Fig. 5. A *push* is generated ahead of the tank.

We can now state the push algorithm as follows:

Algorithm : *Push*

- When a sensor detects a threat it determines the projected velocity vector for the threat k. Let s denote the speed of the threat.
- Let τ_k represent the time specification used by the sensor to push information for the threat. The value τ_k might be different for different types of threats.
- The sensor (or sensors) send one THREAT_WARNING message to each of the leaders of the blocks that lie within the rectangular area with length $s(\tau_k + \Delta\tau)$ and width equal to the *strike range* of the threat. The rectangle is oriented in the direction of motion of the threat.
- Each leader receiving the THREAT_WARNING message broadcasts it within their groups. The THREAT_WARNING message includes the nature of the threat, velocity, and any other information (e.g., confidence level of the sensor in projecting the threat's motion).
- Whenever the threat changes direction or moves a distance such that the furthest block warned is less than $s\tau_k$ distance away from the threat, a new push message is sent to blocks that are within $s(\tau_k + \Delta\tau)$ distance of the threat. In Figure 5, after the tank moves through one (or two) blocks in the indicated direction, a new push is generated to cover blocks a and b.

B. Pull Protocol

The push algorithm as described above succeeds in sending threat warnings to nodes that are within τ_k of the threat. However, nodes that are moving towards the threat (and are further away than $s(\tau_k + \Delta\tau)$ of the threat's position) or nodes that are far away but have a large *time/space* specification need to pull the threat information in order to be appropriately warned. For the sake of clarity we will use the example illustrated in Figure 6. Here, the node on the bottom left is moving at some speed s_n and in time t_n it will be in block A. Assume that threats located anywhere within the dotted box can attack nodes located in block A. The algorithm we use for generating pulls is as follows:

Algorithm: *Pull*

- Let a node's *time* specification be t_n (the same algorithm works if we use a node's distance specification). In other words, the node needs to be warned of all threats it considers threatening that it could encounter within time t_n.
- Let the node's speed be s_n. Therefore in time t_n it will be located in a block that is $s_n t_n$ distance away (in Figure 6 this corresponds to block A).
- The node sends a PULL_REQUEST to the leader of the block it expects to be in after time t_n. In Figure 6 this corresponds to block A. The PULL_REQUEST contains the node's velocity vector as well a specification of threats the node needs to be warned about (for example the pull request may only indicate tanks as potential threats).
- The leader from block A sends back all the information it has about threats that will lie within the dotted region of Figure 6 at time t_n hence. It is, however, likely that the leader may only have incomplete knowledge about threats that are t_n time away. In this case it needs to initiate additional pulls as follows:
  - Let s_k^{max} represent the maximum speed of threats of type k.

- The leader sends a pull request to leaders of blocks that are located distance $s_k^{max} t_n$ away (as shown in Figure 6).
- Leaders of these blocks determine if threats they are aware of will lie in the dotted area (Figure 6) t_n time hence. If so they respond with that information to the leader of block A. The leader of block A then forwards this information to the requesting node.

- Finally, all pull requests are **sticky** with a TTL field (Time To Live). This means that once a leader gets a pull request, it holds on to it until the TTL field expires. Meanwhile, if there is any new threat information, it forwards that to the node that generated the sticky pull request. This mechanism ensures that pull requests only need to be sent infrequently.

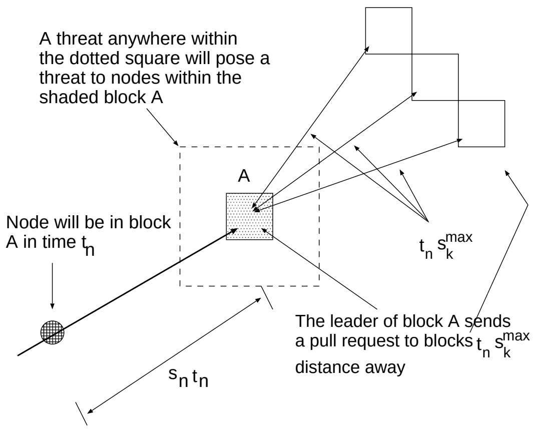

A threat anywhere within the dotted square will pose a threat to nodes within the shaded block A
A
Node will be in block A in time t_n
$t_n\ s_k^{max}$
The leader of block A sends a pull request to blocks $t_n\ s_k^{max}$ distance away
$s_n t_n$

Fig. 6. A *pull* is generated ahead of the node.

C. Leader Maintenance

In order for our protocol to work, we need a leader maintenance protocol as well. This protocol does the following:

- A departing leader selects a new leader in the old block and passes on all the collected threat information to that leader. The identity of the new leader is also broadcast within that block.
- When any node leaves a block, it informs the leader of the old block as well as the leader of the new block.
- If the leader is the last node remaining in a block, when it leaves the block it transmits a message to leaders of all neighboring blocks informing them of all its gathered threat information as well as informing them that the block is now empty. Thus, any pull requests for the empty block will be replied to by one of the neighboring block's leaders (note that pull messages have to traverse a neighboring block to get to the empty block, thus the leader of the neighboring block can respond).
- When a node enters an empty block, it collects threat information from the neighboring blocks and declares itself the leader.

IV. Performance Evaluation

We evaluated the performance of our T-S multicast protocol via extensive simulations. In order to quantify the performance of our protocol, we concentrated on two metrics:

1. *Message overhead:* The metric here is the number of messages/second/node that are exchanged in the course of the simulation run. This includes all push/pull messages as well as control messages exchanged in order to maintain the blocks.
2. *Coverage:* It is possible that some nodes do not receive threat warnings in time. Thus, we measured the percentage of nodes that ought to have been warned of a threat but were not. Ideally we would like this number to be zero. However, periodic network partition and fluctuations of routes make it difficult to achieve this value.

The remainder of this section is organized as follows. We first describe the experimental set up and then describe our experimental results.

A. Simulation Parameters

For the simulation we assume that the battlefield is a 50km square region. The threats consist of tank, shooter and gas threats. The allied forces have tanks and shooters. Tanks move at an average speed of 72kmph, shooters move at an average speed of 9.6kmph and wind speed is constant at 18kmph. The battlefield has sensors capable of detecting tank and gas threats. In the simulations we use 64 gas sensors and 64 tank sensors evenly distributed throughout the batlefield. A gas sensor can detect a gas threat within 1km and a tank sensor can detect tank threats within 1km as well. Shooter threats can only be detected by other (friendly) shooters. Thus, it is possible that a shooter threat may go undetected because of human error. We use a probability of 0.2 that a shooter threat will not be detected. Finally, we assum that tanks, shooters and sensors have radio capability. The transmission radius of the sensors is assumed to be 2km, tanks have a transmission radius of 5km and a shooter can transmit to a distance of 1km. The data rate available is 1Mbps.

The time-space parameters we use were the following. Tanks needed to know about tank threats within 6km and shooters needed to know about tank and shooter threats within 900m. Each simulation is run for 300 seconds of real-time (a couple hours of simulation time) and the simulation time step is 5 msec of real-time. We run each case ten times and compute 95% confidence values. In each case the, the confidence intervals were very tight (less than 5% of the point values).

B. Discussion of Results

In all the following plots we use a constant number of enemy threats. Specifically, we use 10 tank, 10 gas and 30 shooter threats. In Figure 7 we plot the total message overhead as a function of the number of friendly tanks (no shooters) and in Figure 8 we plot the message overhead as a function of the number of shooters (no tanks). First, it is noteworthy that smaller block sizes result in greater message overhead. This is because the number of responses to a pull are greater (one per block) and the overhead of maintaining blocks is also greater (since the time a node spends in a smaller block is smaller). The second interesting observation is that the message overhead increases sub-linearly with increasing numbers of friendly nodes. This indicates that our T-S protocol is scalable to large networks. Intutively this makes sense since the extent of a sensor's multicast is geographically limited to nodes in immediate danger. Thus, even if the network size grows, the message overhead ought ot remain the same.

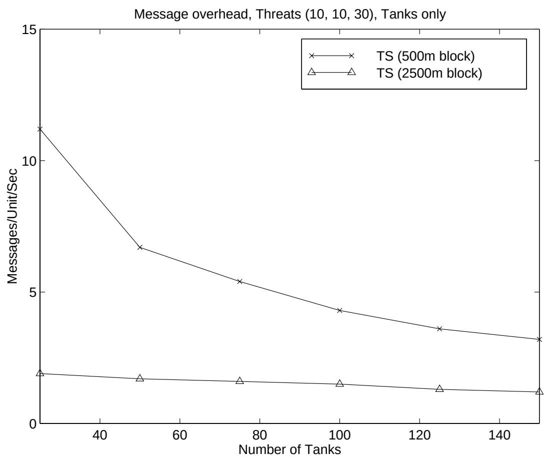


Fig. 7. Message overhead – Tanks only.

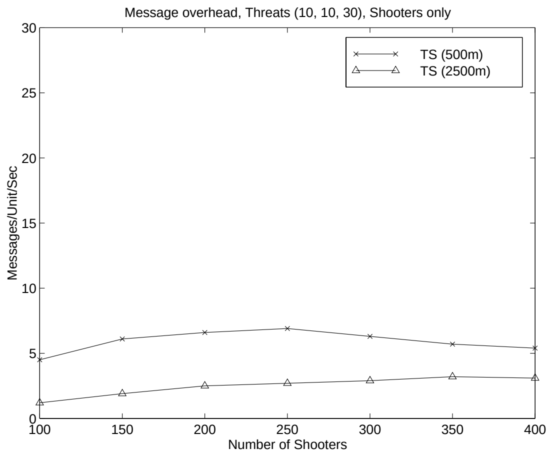


Fig. 8. Message overhead – Shooters only.

Figure 9 plots the success rate as a function of the number of tanks. It is noteworthy that with small block sizes the percentage of nodes warned in time is only about 65% for the case when there are 25 tanks. This number changes to 96% with a block size of 2.5km. Thus, using larger block sizes is good because the message cost is lower and the percentage of nodes warned is higher. Interestingly, the success rate does not appear to vary much with block size in the event that we only have shooters (Figure 10). The explanation for this is that pull messages do not always reach the desired blocks because the network does not have a geographically direct path (we use geographical routing to send messages so if there is no next hop in the direction of the destination, the packet is dropped). However, as the number of shooters increases, the probability of finding a path increases and hence the percentage of nodes warned in time also increases. In the case of tanks, their high speed and greater transmission radius ensures that there is a greater probability of finding routes.

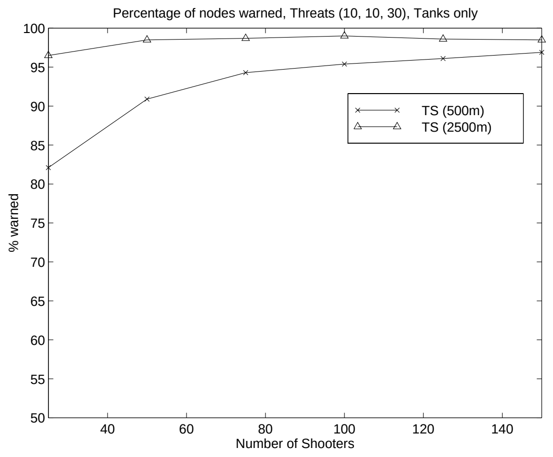


Fig. 9. Success Rate – Tanks only.

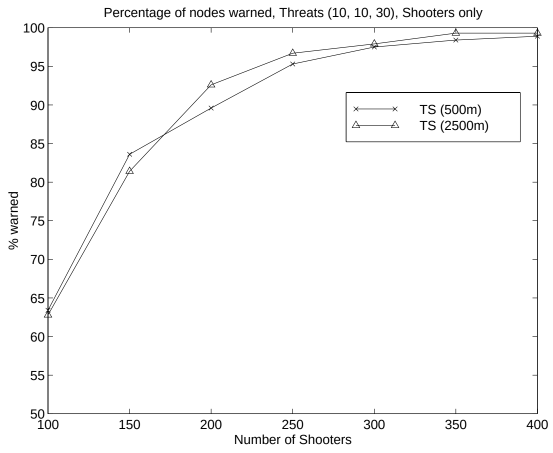


Fig. 10. Success Rate – Shooters only.

V. Future Work

We are extending our simulation to take into consideration the path loss models of the terrain (as this affects radio connectivity), unpredictability of threat mobility, mobility models of nodes that better reflect the way troops are deployed in battle, reliability issues where the information gathered by sensors may not always be trustworthy. In addition, we are developing analytical models that will allow us to better understand the energy/message efficiency tradeoffs in this domain.

Acknowledgements

This work was supported by ONR grant 9910499.

References

[1] E. Bommaiah, M. Liu, A. McAuley and R. Talpade, "AMRoute: Ad Hoc Multicast Routing Protocol", *Internet-Draft*, draft-talpade-manet-amroute.txt, Aug. 1998, Work in progres.

[2] J.J. Garcia-Luna-Aceves and E.L. Madruga, "A multicast routing protocol for ad hoc networks", *Proc. IEEE Infocom'99*, New York, NY, Mar. 1999, pp. 784-792.

[3] S.-J. Lee, M. Gerla and C.-C. Chiang, "On-Demand Multicast Routing Protocol", *Proc. IEEE WCNC'99*, New Orleans, LA, Sept. 1999, pp. 1298-1304.

[4] C.W. Wu, Y.C. Tay and C.-K. Toh, "Ad hoc Multicast Routing protocol utilizing Increasing id-numberS (AMRIS) Functional Specification", *Internet-Draft*, draft-ietf-manet-amris-spec-00.txt, Nov. 1998, Work in progress.