Multiple Simultaneous Acoustic Source Localization in Urban Terrain

Ákos Lédeczi, Péter Völgyesi, Miklós Maróti, Gyula Simon, György Balogh, András Nádas, Branislav Kusy, Sebestyén Dóra and Gábor Pap
Institute for Software Integrated Systems
Vanderbilt University
Nashville, TN, 37235, USA

***Abstract*—Experiences developing a sensor network-based acoustic shooter localization system are presented. The system is able to localize the position of a shooter and the trajectory of the projectile using observed acoustic events, such as the muzzle blast and the ballistic shockwave. The network consists of a large number of cheap sensors communicating through an ad-hoc wireless network, which enables the system to resolve multiple simultaneous acoustic sources, eliminate multipath effects, tolerate multiple sensor failures while providing good coverage and high accuracy, even in such challenging environment as urban terrain. The paper describes the hardware and software platform developed for this application and summarizes the lessons learned during the development of the system.**

Keywords-wireless sensor network; sensor fusion; acoustic source localization; time synchronization; message routing;

I. INTRODUCTION

A common feature of existing countersniper systems is that they contain only a few sensor units deployed around or near a protected area. Our alternative approach was made possible by the emergence of wireless sensor networks. In contrast to current solutions, our system contains a large number of inexpensive sensors. Thus, the system can provide much better coverage and it is robust against multipath effects prevalent in urban environments. Furthermore, our system can resolve multiple, simultaneous shots which no other existing system can do. In addition to these technical advantages, a sensor network-based approach also have several other favorable properties in military operations: speed and ease of deployment in hostile environments, independence of potentially unreliable civilian infrastructure (power and communication network), and robustness against node failures.

Figure 1. depicts the acoustic events generated by a typical rifle shot. The muzzle blast produces a spherical wave front, traveling at the speed of sound from the muzzle of the gun. The shock wave is generated in every point of the trajectory of the supersonic projectile producing a cone-shaped wave front (assuming the speed of the bullet is constant). The angle of the shockwave cone is determined by the ratio of the speed of the projectile and the speed of sound. The sensors autonomously detect the shockwave and/or the muzzle blast, measure their times of arrival (TOA), and send the measured results to a base station through the ad-hoc wireless sensor network (WSN). A fusion algorithm running on the base station determines the location of the shooter and the trajectory of the projectile. The sensor network utilizes several middleware services to maintain communication between the base station and the sensors, synchronize the clocks of the sensors, and perform self-localization.

The DARPA/IXO NEST program (F33615-01-C-1903) has supported the research described in this paper.

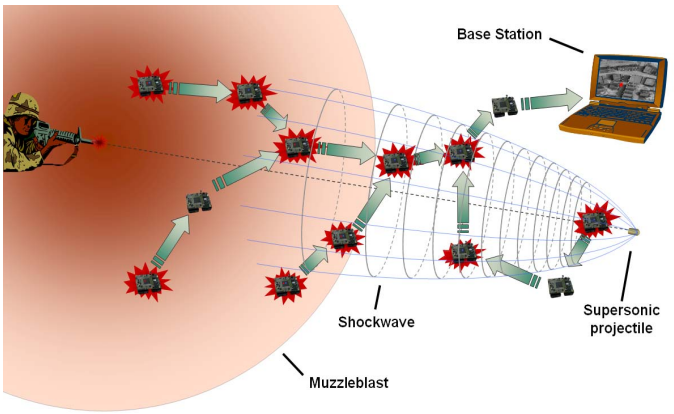

Figure 1. System concept

The 3D localization accuracy of the developed prototype system is 1 meter on average. The latency of a sixty-node six-hop setup is below 2 seconds. Furthermore, the system can accurately localize several shots per second. This rate is limited primarily by the low communication bandwidth available on the current sensor network platform.

We reported on the first generation of the system in [1]. Time synchronization and message routing was discussed in detail in [2] and [3], respectively. The aim of this paper is to share the many lessons we learned during the development of this successful sensor network application. In addition, we describe the new techniques we applied in the second generation system including a new sensor board, new detection algorithm, a novel time synchronization approach and a new shockwave fusion algorithm. The remainder of the paper is organized as follows. In the next two sections we describe the hardware and software platform we developed and applied. Then we briefly present and evaluate the results. Finally, we detail the most important lessons we learned.

II. HARDWARE PLATFORM

The hardware platform is built upon the UC Berkeley MICA2 mote device running the TinyOS embedded operating system [6], a widely used component-based architecture targeting wireless sensor network applications. Open interfaces at the software and hardware levels made it possible to integrate specialized smart sensor elements and supporting middleware services. Each MICA2 mote is furnished with an ATmega 128L 8-bit microcontroller with 128 Kbytes instruction memory, 4 Kbytes data memory and typical embedded

peripherals built in. The on-board radio transceiver operates in the 433 MHz ISM band and has a maximum transfer rate of 38.4 Kbits/sec with the maximum range of about 300 feet.

Real-time detection, classification and correlation of acoustic events require processing power and buffer sizes not present in standard microcontroller-based embedded devices. To overcome these limitations, application-specific sensor boards have been designed and built at Vanderbilt University. The different architectures reflect the current dilemma faced by many signal processing engineers today.

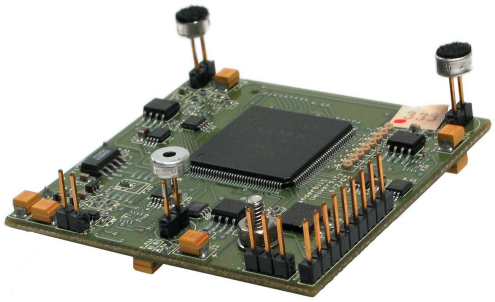

Figure 2. FPGA acoustic sensor board

Our first design (see Figure 2.) utilizes a Xilinx XC2S100 FPGA chip with three independent analog channels exploiting the inherent parallelism of the hardware. The algorithms – implemented in VHDL – are focusing on time domain analysis of acoustic signals captured at high sample rates (1 MSPS). Hardware and software interfaces (I2C bus, interrupts, led display and serial A/D) are implemented as custom IP cores in the same gate array. While this approach completely avoids the instruction-fetch, load/store bottlenecks of traditional processor architectures and provides efficient resource utilization, the size of the FPGA component severely constrains the complexity of our algorithms. Suboptimal power consumption of the processing unit is another handicap in the sensor network domain.

To overcome these limitations another sensor board has been developed, where customized analog signal paths and an energy-efficient, powerful DSP processor make the unit uniquely suitable for power constrained applications. At the heart of our second platform (Figure 3.) is a low-power fixed point ADSP-218x digital signal processor running at 50Mhz. Its internal program (48KB) and data (56KB) memory buffers with advanced addressing modes and DMA controllers enable sophisticated signal processing and advanced power management methods.

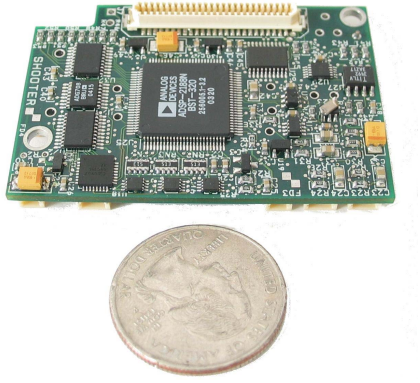

Figure 3. DSP-based acoustic sensor board

Two independent analog input channels with low-cost electret microphones pick up the incoming acoustic signals utilizing a 2-stage amplification with software programmable gain (0-54dB). The A/D converters sample at up to 100kSPS at 12-bit resolution. Analog comparators with software adjustable thresholds can be used to wake up the signal processor from low-power sleep mode, enabling continuous operation for weeks on two AA batteries.

The signal processing board can be used as a stand-alone device or connected to other embedded hardware using standard and custom communication interfaces. Programmable interrupt and acknowledgement lines and a standard I^2C bus connection enable integration with the MICA2 mote [6, 8]. The board also provides a standard asynchronous serial interface, perfectly suited for PC, laptop or PDA connections.

The combination of computational power, energy efficiency and specialized circuits made it possible to utilize more sophisticated frequency and time domain analysis detecting and classifying acoustic events more precisely.

The FPGA and the DSP boards running the detection algorithms continuously draw 30 and 31 mA, respectively. While we have not implemented the power saving mode on the DSP board yet, we expect this number to drop to 1-5mA in various sleep modes. For comparison, the Mica2 mote draws 15 mA running the countersniper application.

III. SOFTWARE PLATFORM

An earlier version of the system was described in detail in [1]. Here we present a summary of the software architecture (Figure 4.). The Muzzle Blast and Shockwave Detector is implemented in VHDL on the FPGA of the first generation sensor board and in C on the DSP of the new board. The TOA data from either board is sent through the I^2C interface to the mote. The Acoustic Event Encoder assembles a packet containing the TOA data and passes it to the Message Routing service.

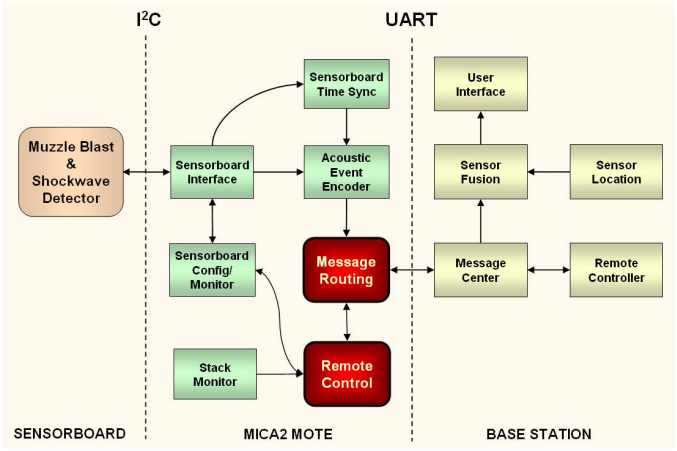

Figure 4. Software Architecture

In addition to transporting the packets to the base station through multiple hops, the Message Routing service also performs implicit time synchronization. Additional software components running on the mote include a Remote Control service enabling the configuration/polling of a single node, a group of or all of the nodes from the base station. A Stack Monitor makes sure that the limited memory of the mote is not exhausted.

The Base Station runs the Sensor Fusion algorithm utilizing the known sensor positions and displays the results on the User Interface. The accuracy and/or range of existing sensor self-localization methods (including our own [4]) are not satisfactory for the shooter localization application. Hence, up-till-now all tests of the system were performed utilizing hand-placed motes on surveyed points.

The next three sections summarize the key components of the system.

A. *Detection*

The first version of the sensor board was designed with three independent acoustic channels allowing on-board Angle of Arrival (AoA) estimation using time of arrival (ToA) estimates of the three channels. Since the microphones of the individual channels were located two inches apart, the resolution and accuracy requirements were in the microsecond range. Hence, the incoming signal is sampled at 1MHz and then it is compressed using Zero-Crossing (ZC) coding. ZC coding provides large compression rate enabling real-time processing in the latter stages of the signal processing chain. Although the coding is lossy, those features of the signal between zero crossings that are necessary to achieve high precision ToA estimates are preserved: start time, min/max amplitude, length, rise time, and previous average amplitude. The ZC-coded signal is used to detect possible occurrences of shockwave and muzzle-blast patterns.

The signal processing algorithm running on the second generation board uses time-frequency analysis combined with time-domain feature extraction. In the first stage, possible events are detected using the signal's energy distribution in the frequency domain. This stage has a high false positive ratio, but successive stages eliminate false candidates based on the changes of the energy distribution in time. The next stage utilizes a signal-to-noise-ratio-like quantity (SNR), calculated as the ratio of the signal energy in certain frequency ranges of interest. The amplitude, slope and width of the SNR peaks are used to eliminate the majority of false positives. The next stage determines the exact starting point of a candidate in time domain, and further tests involving the shape of the signal are performed. The candidates are then time-stamped, and a quality descriptor (QD) is assigned to them. The QD contains confidence values, describing the 'muzzle blast'-ness or 'shock wave'-ness of the detected event. Based on the QD, either a local decision is made whether the event is to be reported at all, or the decision is left to the central fusion algorithm, where the time-stamp along with the QD is used to determine the location of the shot.

Both of the boards and the detection algorithms worked well under a variety of circumstances. Depending on the type of gun and ammunition, the muzzle blast detection range was between 30 and 150 meters. The shockwave detection range is between 30 and 50 meters. Note that this is not as critical because if the projectile does not go the near the sensor field, then the shot is not interesting from an application standpoint. There were hardly any false positives for either muzzle blast or shockwave. Vehicle noise, engine backfire, training grenades and regular urban noise did not cause any detections. The only false detections were due to raindrops hitting the microphone directly. As these are events limited to a single microphone at a time, light to moderate rain did not affect system performance at all, because the sensor fusion immediately determined that no single source could have produced the events within a few tenths of a second.

B. *Routing-Integrated Time Synchronization*

The Message Routing service utilizes the Directed Flood Routing Framework [3]. However, in addition to the sensor reading, a radio message includes an age field, which contains the elapsed time since the event detection. Each intermediate mote measures the elapsed time from the reception of a data packet till its retransmission. The age field is updated upon transmission using a precise time stamping method described in [2]. When the sensor reading arrives at the destination, the age field contains the sum of the offsets measured by each of the motes along the path. The destination node can determine the time of the event by subtracting age from the time of arrival of the message. In essence, the clock of the base station becomes the global clock. The average accuracy of this approach implemented on MICA2 motes is tens of microseconds depending on the number of hops and other factors. As the speed of sound is approximately one foot per millisecond, this time synchronization error may cause a shooter localization error that is well under an inch in the worst case.

C. *Sensor Fusion*

The sensor network delivers the measured shockwave and muzzle blast TOA data to the base station after each detected shot. The data set contains correct measurements that are detections of primary (line-of-sight) acoustic events, and they may also contain erroneous data, typically due to multipath effects. Multiple shots may occur at different locations, but close in time, resulting in mixed measurements containing correct measurements of multiple shots and also erroneous data. The task of the sensor fusion algorithm is to estimate the shooter location and the trajectory of the projectile, in spite of the possibly large number of incorrect measurements. Furthermore, it has to deal with two more sources of error: (1) imprecisely known sensor locations and (2) time synchronization error.

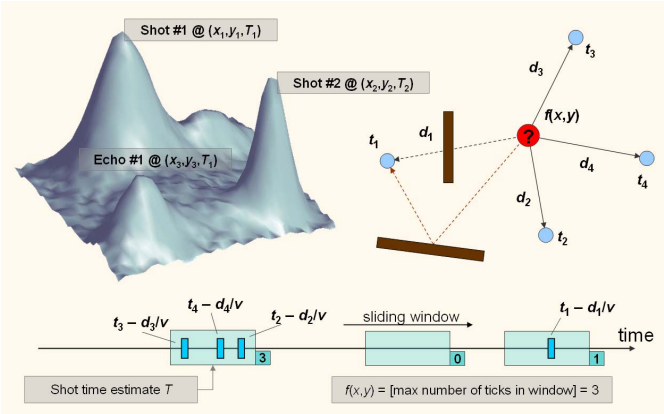

Figure 5. Muzzle Blast Sensor Fusion

The traditional Time Difference of Arrival (TDOA) approach is not able to handle either multipath effects or multiple simultaneous shots. Our fusion algorithm is based on a search on a surface defined by a consistency function. The global maximum of the surface defines the estimated shooter position. Multiple shots are shown as multiple local maxima on the surface. A formal definition of the consistency function and the search to find its maxima are given in [1]. Here we present a conceptual description.

Consider the right hand side of Figure 5. The four blue circles represent four sensors that detected a shot. Let us assume that the shot was fired from the red position. Since we know the sensor positions and the speed of sound, we can plot the timeline of events when the shot had to be fired according to the sensor readings (bottom of Figure 5.). If the position is the correct shooter position, then all line-of-sight sensors will agree on the time of the shot. That is, these detections will fall within a narrow window whose width is determined by the possible detection errors that are dominated by the sensor position errors. So, if we know the sensor positions to one foot of accuracy then all line-of-sight detections will fall within a millisecond wide window on the timeline at the correct shooter position. Non-line-of-

sight detections, on the other hand, will show up as outliers. Consider the first sensor in Figure 5. whose detection time was t_1. The sound of the shot traveled longer then the distance we use in our computation, hence, it will result in a larger estimated shot time.

The consistency function is defined for every point in the three dimensional area of interest as the maximum number of sensor detections that fall within a narrow time window. The estimated shot position will be the global maximum of this function. Note that multiple shots will show up as multiple peaks. Also note that a consistent echo, that is, an echo from the same obstruction measured by multiple sensors, shows up as a local maximum also. However, it can be shown that the estimated absolute time of a real shot and a corresponding consistent echo are the same. As long as more sensors detect the real shot than the echo, peaks due to echoes can be easily eliminated. This assumption proved to be realistic in our field experiments.

A multi-resolution search procedure based on interval arithmetic finds the global maximum rapidly [1]. Sensor readings contributing to this position are then removed from the list of unclassified acoustic events and the procedure is repeated to find subsequent shots.

The above description applies to muzzle blasts only. Recently, we have generalized the approach to shockwaves as well. In this case, three more dimensions are added to the original four dimensions (x, y, z spatial coordinates and the shot time). These are the azimuth and elevation of the projectile trajectory (estimated as a line) and the speed of the bullet (estimated as constant). However, the search in this 7 dimensional space proved to be computationally infeasible using a current desktop class computer. Instead, we applied a genetic algorithm-based approach to determine projectile trajectories. [10] provides a detailed description of the approach.

IV. RESULTS

To evaluate the performance of the shooter localization system, multiple field experiments were conducted in US Army Military Operations in Urban Terrain (MOUT) facilities. A typical setup utilized 60 motes deployed in the central area of the facility covering an approximately 100m by 50m area with mean node spacing distance of 5m.

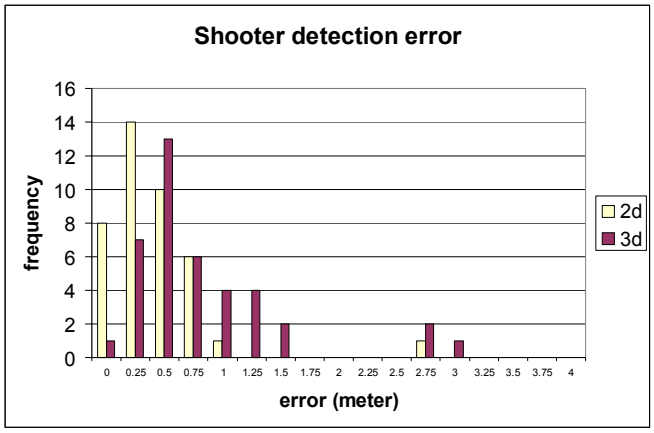

Figure 6. Experimental results

In one particular test scenario, 40 shots were fired of which 18 were blanks and 22 were short range training ammunition (SRTA). Since the performance of the system was equivalent for both types of ammunition, only the unified results are presented. In the following summary, we refer to (x,y) localization error as 2D error (elevation is not considered), and (x,y,z) localization error as 3D error. Figure 6. shows the histogram of the 2D and 3D errors for the set of test shots. The position error (in meters) and the number of shots are shown on the horizontal and vertical axes, respectively. The average 2D error was 57cm, while the average 3D error was 98cm. These numbers are exceptional.

As one can see, a single shot had more than one meter 2D error, while two more had 3D errors in the 3-meter range. One of these shots was taken near the edge of the sensor field and only a handful of sensor detected it. The other two created a relatively flat consistency function around the true location possibly due to echoes. In general, the 3D accuracy is worse than the 2D because most of the sensors were on the ground and the vertical resolution of the system, therefore, was not as good.

These results were obtained utilizing only the muzzle blast information in the sensor fusion. Recently, we conducted our last field experiments to evaluate the shockwave-based sensor fusion for long range shots. We applied a genetic algorithm-based shockwave fusion [10] and a novel range estimation method [10] when some muzzle blast detections were also available. The results were very encouraging. In one particular test, 12 shots were fired through the middle of the network, so there where sensors on each side of the trajectory. The shooter was approximately 100 meters from the edge of the sensor field. The average azimuth error was 0.66 degrees, the average elevation error was 0.61 degrees, and the average range error was 2.56 meters. In another test, 11 shots were fired from the same distance near the edge of the network, so there were no or only a few sensors on one side of the trajectory. The average error increased to 1.41 degrees in azimuth, 1.11 degrees in elevation and 6.04 meters in range. These numbers are comparable to those of existing centralized countersniper systems.

Utilizing the test data, we performed simulations to check the sensitivity of the system to sensor localization error. Surprisingly, we found that the trajectory estimation accuracy is largely insensitive to sensor positions. Even when an extra 2m random error was added to the sensor positions, the azimuth error still remained below 2 degrees. For more details, refer to [10].

V. LESSONS LEARNED

Developing a non-trivial application on a resource-constrained sensor network platform has taught us many lessons. The most important one is that there is a very long way from an algorithm on paper to a middleware service on the target hardware platform working under real-world environmental conditions. What are the special characteristics of this computing field that make it especially difficult?

A. Moving Target

The field of wireless sensor networks is a relatively new domain characterized by rapid innovation. As such, the available hardware and software platforms evolve at a fast pace. Developing applications for a moving target is challenging. For example, the MICA2 mote, the successor of the original MICA, became available in early 2003. Among other things, it had a new radio chip and required a new radio stack implementation in TinyOS. It turned out that collision avoidance did not work perfectly, which significantly degraded the communication bandwidth. Some of our existing middleware services had poor performance on the new platform, which led us to the development of algorithms that worked well even in the presence of collisions.

TinyOS itself is an exemplar of a very successful, but rapidly evolving system. Version 1.0, which came out in the fall of 2002 and was a complete rework implemented in a new language, provided great improvements, but required a significant effort in porting all

existing services and applications. We found that one has to balance the stability of the target application and the need of importing parts of the latest revisions of the TinyOS development tree.

There is a rich body of work documented in the sensor network literature. However, few proposed algorithms are ever implemented on hardware. Those with available implementation become outdated very quickly unless the authors remain vigilant about porting their code when the hardware and software platforms change. Even then, it is not guaranteed that the algorithm works just as well on the new platform, as it did on the old one. These factors hinder software reuse significantly.

B. Integration

Middleware services do not work in isolation. Any meaningful application uses multiple middleware services and other application components. They must interact and share the limited resources of the platform. One of our bugs was traced to the stack overwriting statically allocated memory under some rare conditions. This led us to the development and religious use of a stack monitoring service. In addition to the hardware limitation of 4 kilobytes of available data memory, TinyOS itself can run out of resources, such as timers and tasks. This has also occurred several times.

All these observations point in one direction. Middleware services must be kept as simple as possible. Simple algorithms typically need fewer resources and have fewer interactions that are easier to comprehend and hence, debug. This means that general purpose protocols that are meant to support a variety of applications are usually too heavyweight for the mote platform. In our experience, the development of application-specific middleware is the rule rather than the exception. For example, there is no message traffic in our network until a shot is fired. Then a lot of motes need to send messages to the base station almost simultaneously. As many messages as possible need to be delivered in the first second, but it does not necessarily matter which ones. We developed a simple routing protocol specifically optimized for this set of requirements instead of taking an existing approach, porting it to the then new MICA2 platform and hoping that it'll perform well in our application.

C. Simulation

Embedded systems in general are characterized by their tight integration with the physical world. This is especially true for wireless sensor networks, since their primary purpose is to measure some physical phenomenon, they must operate in an uncontrolled environment, and they use wireless communication. These facts limit the applicability and usefulness of simulation. Few simulators take the physical world into account. For wireless networks that is usually limited to the simulation of the radio channel. But how do RF multipath effects prevalent in urban environments or draining batteries effect radio range? Some of the complex simulator frameworks (e.g. ns-2) might be able to be configured to answer some questions like these, but the effort required to set up realistic simulations is very high.

The current state-of-the-art in mote simulation presents other problems too. TOSSIM, the TinyOS simulator, takes the actual code the motes run, but its radio model is very simplistic. More realistic simulators [7, 9], on the other hand, require reimplementation of the application in their language using their API. Such simulation can be very helpful in the early phases of algorithm development, but does not help in debugging the actual code. These difficulties and the tight deadlines made us do hardly any simulation in the development process.

D. Development cycle

In many ways, developing code for the mote platform is reminiscent to the earlier days of computing when memory usage or the number of floating point operations in a program were important considerations. When you have 4 kilobytes of data memory and twenty some useful bytes in a message, literally every bit counts again. And you need to (re)learn that you do not multiply or divide, only shift.

Another similarity to the past is the speed of the debug cycle. Reprogramming a hundred nodes by plugging them in the programming board one by one is not a speedy operation. Thus, we needed to make radio reprogramming work somewhat reliably for at least the one-hop case. The primary debugging tool we used to have was the three LEDs on the mote. Instead, we created a tool that works like "printf," routing the data back to the base station and displaying the text there. In this sense, basic tool development is an integral part of application development just like in the past.

Another unique feature of sensor network applications is that a small change in the code or a minor new feature will have an impact on many parts of the overall application. The primary reason for this is the fact that there are typically three or more different hardware components working together in a typical WSN application: the sensor board, the mote and the base station. For example, when we wanted to add a simple data recorder, so that we can analyze the signal shapes the acoustic channel records on the sensor board, we needed to modify the sensor board code. Then we needed to add the capability on the mote side to receive a large data buffer from the sensor board through the I^2C interface, store it, break it up into small packets and send it to the base station using the message routing service. Furthermore, the base station Java code needed to be modified to receive the packets, assemble them in the correct order and request any missing packets from the sender. Finally, the pieces needed to be integrated and tested. A relatively simple new feature took a couple of manweeks to fully implement and verify.

E. Testing

Testing may be the most time consuming task in putting together a real world sensor network system, especially, if it is an outdoor application. The main problem is that what works using a handful of nodes in the lab will almost always not work under real deployment conditions. Differing conditions include communication distances (and hence, different topology and network diameter), multipath effects, weather conditions (temperature, precipitation), noise and other miscellaneous environmental factors. And during a field test you typically try out things that you do not think of in the lab. For example, you may put the motes in tall grass and realize that the effective communication range is cut in half.

Therefore, the system needs to be tested under real deployment conditions frequently. In our case that means loading 50 motes and sensor boards with the current software, changing the batteries, and taking them along with laptops, rifles and ammunition to a farm outside of town. Once there, the mote positions need to be surveyed to better than 1 foot accuracy, a time consuming task in itself. The whole setup needs a handful of people and takes several hours even before you start testing. At this point, you do not want to realize that because of a trivial error, the system does not really work and you have to postpone the test. Therefore, before a field test, you need to test every piece of the system and the whole application extensively in the lab. You need to find all the problems you possibly can in the lab before committing yourself to an expensive field test. Therefore, a complete field test of the system can take multiple people several days.

F. Scalability

The necessary sensor density and the size of the area that can be covered depend on many factors including the density of the urban area, overall accuracy requirements and the expected lifetime of the system. We estimate that scaling up to system to 100 nodes deployed in a typical urban area and covering 200x200 meters could provide the accuracy and latency presented in this paper. Beyond such deployment area and number of nodes, the network connectivity and the latency would become issues. However, this area is hardly more than one percent of a square mile. A realistic military application would require the coverage of several square miles necessitating orders of magnitude larger network size.

While this particular system has latency and deployment density requirements that may be more severe than most WSN applications, the limited radio range will constrain the deployment area of any sensor network application. As many others have observed, this mandates a hierarchical network architecture where groups of motes are clustered around more powerful nodes. The main requirement is that this "second layer" node has a different communication channel to other such "supernodes" providing higher bandwidth and longer range. While there are several attempts at creating such hierarchical network architecture, significant challenges remain. In particular, the power requirement of such supernodes mandates very large batteries or results in severely limited lifetime.

CONCLUSIONS

The most important lesson we learned is how wide the gap is between an algorithm on paper, or even performing satisfactorily in a simulator, and working on the actual hardware as an integral part of a complex application deployed in the field. The tight integration of WSNs and the physical world, the severe resource constraints and the wireless communication are the most significant factors responsible for this gap.

ACKNOWLEDGEMENTS

The authors would like to thank Vijay Raghavan, Keith Holcomb, Al Sciarretta, Tony Mason, David King, Tamás Lédeczi, Béla Fehér, Janos Sztipanovits, and Ben Abbott for their valuable contribution to this work. This work would not have been possible without the help and dedication of the people at the US Army Dismounted Battlespace Battle Lab at Ft. Benning and the U.S. Army Aberdeen Test Center. We also thank the anonymous reviewers for their constructive comments.

REFERENCES

[1] Simon, G., et al.: Sensor Network-based Countersniper System. In *Proc of the Second ACM Conference on Embedded Networked Sensor Systems (SenSys)*, November 2004.

[2] Maroti, M., Kusy, B., Simon, G., Ledeczi, A. The Flooding Time Synchronization Protocol. In *Proc of the Second ACM Conference on Embedded Networked Sensor Systems (SenSys)*, November 2004.

[3] Maroti, M.: The Directed Flood Routing Framework. In *Proc of the 5th International Middleware Conference*, October 2004.

[4] Sallai J., et al.: Acoustic Ranging in Resource-Constrained Sensor Networks, In *Proc of ICWN '04*, Las Vegas, NV, June, 2004.

[5] Gay, D. et al.: The nesC Language: A Holistic Approach to Networked Embedded Systems, In Proc. of *PLDI*, 2003

[6] Hill, J., Culler, D.: Mica: A Wireless Platform for Deeply Embedded Networks. *IEEE Micro*, Vol. 22, No. 6, 2002, pp. 12–24.

[7] http://www.isi.edu/nsnam/ns/

[8] http://www.xbow.com/

[9] Simon G., et al..: Simulation-based optimization of communication protocols for large-scale wireless sensor networks, *In Proc of 2003 IEEE Aerospace Conference,* March, 2003.

[10] Balogh, G., et al..: Wireless sensor network-based projectile trajectory estimation, Technical Report, ISIS-05-601, February 2005, http://www.isis.vanderbilt.edu/projects/nest/applications.html