A File System Abstraction for Sense and Respond Systems

Sameer Tilak[1], Bhanu Pisupati[2], Kenneth Chiu[1], Geoffrey Brown[2], Nael Abu-Ghazaleh[1]
[1]*Computer Science Department, State University of New York (SUNY) at Binghamton*
[2]*Computer Science Department, Indiana University*

***Abstract*— The heterogeneity and resource constraints of sense-and-respond systems significantly challenge system and application development. In this paper, we present a flexible, intuitive file system abstraction for organizing and managing sense-and-respond systems based on the Plan 9 design principles. A key feature is the support of multiple views of the system via filesystem namespaces. Constructed logical views present an application-specific representation of the network, thus enabling high-level programming. Concurrently, structural views of the network enable resource-efficient planning and execution of tasks. We present and motivate the design using several examples, outline research challenges and our plan to address them, and describe the current implementation state.**

I. Introduction

The heterogeneity and resource constraints of typical sense-and-respond (S&R) systems pose daunting challenges to system and application development. These challenges are further exacerbated by the lack of simple abstractions for the use and development of these systems. In this paper, we show how the principles of Plan 9 [1] can be applied to S&R systems, resulting in flexible, intuitive systems supporting multiple logical views. Applications can then use the view with the most appropriate organization and abstraction.

Sense-and-respond systems typically comprise a diverse set of hardware and software elements. Hardware elements include a wide variety of different sensor and actuator types, ranging from COTS to highly-specialized, one-of-a-kind parts. Software elements draw from numerous domains, including the natural sciences, artificial intelligence, sensor networks, and embedded systems. Further increasing the diversity is the various ways in which the software and hardware elements may interact, such as event-driven, polled data, or data streams. This heterogeneity greatly complicates the development of reliable, effective S&R systems.

A crucial component of many S&R systems is wireless sensor and actuator networks. These networks promise to revolutionize sensing across a wide range of civil, scientific, military, and industrial applications. For example, thousands of sensors can be deployed across the landscape to monitor for chemical and biological threats, to monitor for interesting ecological events in migration patterns [2], or to track a smoldering forest fire for conditions that might lead to an outbreak. Responses may range from alerts to the use of actuators to mitigate the damage.

This work was partially supported by NSF Award SCI-0330568, NSF Award DBI-0446298, and NSF Award EIA-0202048.

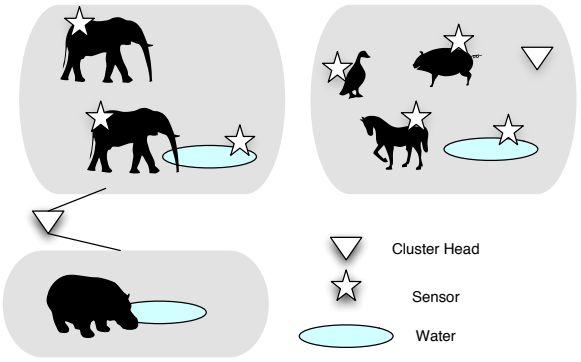

Fig. 1. An example wireless sensor network in a zoo. Sensors track animal locations and resources such as food and water. The network is divided into two clusters, each consisting of a cluster head.

The inherent resource constraints of WSNs pose significant challenges to this vision, however. Wireless sensors are typically limited in power, weight, and size; and communication is often unreliable. These constraints further exacerbate the problems created by the heterogeneity of S&R systems.

Successfully addressing these multi-dimensional challenges relies crucially on developing an effective abstraction for sensor networks. A simple and well understood abstraction can significantly ease both system development and application development. Many sensor networks are deployed by scientists and researchers whose domain of expertise is not computer science. We believe that providing these scientists with a simple and intuitive interface to program, access, configure, and debug sensors can facilitate deployment of large scale sensor networks to a great extent.

Motivated by this need, we propose a simple yet powerful filesystem-based abstraction of sensor networks based on Plan 9, which espoused that the file system metaphor (as seen, for example, in the `/proc` file system) can be adopted for almost all aspects of system design and development. Not only can files be used to store a named sequence of bytes, but also to replace many aspects of communication and control that are typically performed using system calls. A key feature of our proposed solution is the ability of the application to define multiple namespaces to organize the sensor

network in an application specific manner. Another advantage is that we can now exploit, perhaps with some adaptation, much of the work in distributed file systems, such as Coda [3] to address various systems issues such as consistency models.

Another commonly proposed abstraction of WSNs is that of a database [4]. Typically, these databases present application-level information, isolated from the resources providing the data. Upstream data acquistion and processing then lacks resource knowledge, precluding the application of the end-to-end principle and complicating efficient implementations. Infrastructure services also cannot be built on the database abstraction, since by nature these require low-level resource information. In contrast, by providing logical and structural namespaces, our file system abstraction can present information at a wide variety of levels, making it suitable both for building applications and infrastructure.

We model a sensor network as a set of clusters, each with a cluster head. Cluster membership is normally determined geographically. Our model is intended merely to provide a concrete basis for demonstrating the utility of our file system abstraction, and not as an end unto itself. With this abstraction, an application might access sensor data geographically by reading from a path `/location/54W/35N/data`, or logically such as `/data/temperature/snakes`.

This paper contributes a file system abstraction for sensor networks and a proof-of-concept implementation within the ns-2 simulator. The Plan 9 protocol for implementing the file system abstraction, Styx, has already been well-researched on various distributed computing platforms. We thus focus our attention on its implementation in sensor networks.

II. A Filesystem Abstraction of Sensor Networks

The application of file system abstractions to sensor networks is inspired by the tenets of the Plan 9 and Inferno operating systems [1][5], whose defining feature was their uniform treatment of devices and files. This section describes the use of the file system abstraction as a convenient and scalable means to access, configure, and debug sensor networks.

Figure 1 shows a sample sensor network deployed in a zoo, with sensors tracking different animals and resources such as food and water. The network is divided into two clusters each consisting of a cluster head. The sensors themselves may each differ in functionality (temperature/position), hardware type (MSP/AVR), and software platform. Figure 2 shows a typical directory layout for such a network.

The file system representation naturally captures the structure of the network in addition to depicting logical attributes such as aggregation properties and groupings. The root directory `network` encapsulates the whole network. It has a subdirectory for each cluster, each which in turn has three subdirectories named `sensors`, `aggrData`, and `groups`. The `sensors` directory provides direct access to the sensors and has one directory corresponding to each. A lot of sensor network applications are data centric and often however, rather than the individual sensor values, what is of interest is the aggregate value of a property observed at different sensors. These cluster-wide properties can be readily retrieved via the `aggrData` directory, which contains files `avgTemp` and `avgLight`, representing the average temperator and average light readings, respectively. We thus embed "intelligence" into the file system, enabling it to process and interpret data (such as averaging a set individual readings), rather than just storing and presenting it. Finally, the `groups` directories demonstrate the logical grouping of the sensors according to specific criteria. The grouping shown is based on animal type, but could have been based on geographic location of the sensors. (or animals).

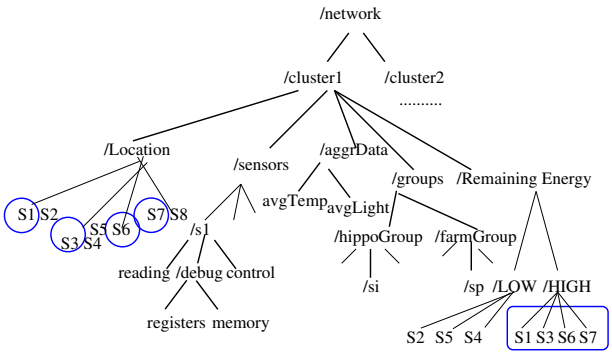

Fig. 2. Namespace for a sensor network.

The task of locating and denoting a sensor device effectively reduces to finding the path for its corresponding file in the namespace. Sensor 1's value, for instance, is read from `/network/cluster1/sensors/s1/reading`. The low level operations inherent in retrieving the values are cleanly hidden away by the uniform file interface, which also conceals heterogeneity among sensors. Furthemore, some sensors represented as part of the network may in fact be simulated while others may be real.

Apart from accessing and reading sensor values, our file system approach also supports configuration and debugging of sensors. The file system can translate writes to the `control` file to control operations on the sensor, such as reset, wakeup, or sleep. The file system may also facilitate debugging by exposing the sensors' `registers` and `memory` as files. An external debugger can then use the file system interface to debug software executing on the sensors. described in the next section.

The file system approach can flexibly partition functionality at different levels of the sensor network, such as at the sensor, cluster head, or client; thus providing a single paradigm even for end sensor devices that may be

computationally lightweight. Logically combining multiple networks now becomes analogous to mounting the networks' file system representations under a common directory.

III. Architecture

As in Plan 9 and Inferno [1][5], all resources are named and accessed as files within a hierarchical directory structure, implemented using the Styx protocol from Inferno[6]. Systems can overlay arbitrarily complex data and sensor policies using multiple simultaneous namespaces, each providing a different perspective of the same physical sensor network.

An entity that wishes to interact with a sensor network mounts its file system and execute the appropriate file operations. The entity implicitly assumes the role of a file system client and correspondingly the filesystem implementation assumes the role of a server. The client and server interact using the Styx messaging protocol to encode the various file operations. Message are always exchanged in pairs, with the client initiating the exchange and the server responding. The client starts a session by connecting to a server using a *Tattach* message. The client may then navigate the directory tree using the *Twalk* message (analogous to the UNIX `cd` command). Other standard operations such as opening, reading, and writing to files are performed using the *Topen*, *Tread*, and *Twrite* messages respectively. These operations may block, but multiple outstanding requests may be issued to compensate.

Client applications do not directly issue Styx messages, but rather use a software library we provide. The interface consists of typical file operations such as *open*, *read*, and *write*. Details of the underlying messaging protocol are completely concealed from the client application.

The file server implementation of sensor networks has two components in its core, namely device-level file servers and multiplexers. Device-level servers reside in the leaf sensors, and define a static directory structure and methods for accessing individual files (really named resources). These fundamental servers provide the most basic interactions with the sensors such as reading the value and primitive control operations. Correspondingly, these servers store minimal dynamic state about themselves and and active clients, and hence require limited runtime memory.

Multiplexers merge different device-level file systems, and would typically reside in the cluster heads to provide a cluster-level namespace. Multiplexers support multiple client connections and multiple outstanding requests. At startup time, the multiplexer engages a discovery process to determine the topology of its associated sensors. It then reads the static directory structure from the device level file systems of all sensors to create cluster-level file system hierarchy. When a client issues a read on the *reading* file inside one of the sensor directories, the multiplexer uses the file descriptor in the *Tread* request to map (multiplex) it to a particular device file system in its namespace. It then reissues the request to that device file system to obtain the sensor reading which is then returned to the client. Multiplexers can receive and process new requests while waiting for a reply from an outstanding request to a device file server.

A multiplexer may also have additional responsibilities, such as data aggregation. For example, an average temperature or position might be provided by by `avgTemp` and `avgPosn` files in the cluster namespace, respectively. Since sensor networks often apply various application-specific filters on the sensor data, the multiplexer implementation allows the aggregation function to be reconfigured on demand through dynamic libraries as described in Section IV

The multiplexer also manages logical groups of sensors shown in the `groups` directory in Figure 2. Sensors are sorted into groups during startup enumeration and also afterwards when new sensors come online.

Multiplexers offer great flexibility in partitioning application, configuration, and debugging functionality between different components of the sensor network. Consider a debugger application that is debugging code executing on a sensor node. The debugger typically requires access to registers and memory on the sensor. Instead of implementing the low-level functionality to retrieve these values in the debugger itself, the functionality can instead be implemented in the device-level servers as files, with the multiplexer in the cluster head providing higher-level organization. The debugger then accesses the registers and memory indirectly by reading and writing to these files, upon which the device-level server performs the necessary low level procedures. As another example, migration from a simulated sensor network to a real network is straightforward. The device file servers can present the same file interface to the application regardless of whether the server is accessing a simulated sensor or a real sensor.

IV. Research Challenges

In this section, we discuss the research challenges specific to using the file system abstraction in a sensor network environment. The following challenges are identified.

Supporting resource efficient operation: Abstraction such as those mentioned in Section II hide complexities of the underlying system, and provide rich and intuitive interfaces to the end user. Since wireless sensors are often resource-constrained, however, the implementation of the file system abstraction must be reasonably efficient; and thus the protocols must be designed to operate within severe constraints on computational power, energy, and storage. We selected the Styx protocol in part because it is lightweight, and does not impose excessive overhead, as detailed in Section VII.

Application-level operations must also be resource-efficient, and so we propose the construction of a standard resource namespace that exposes resource infor-

mation to the application, such as the available energy or storage space on a given sensor node. An example of resource-efficient query execution is presented in Section VI.

While the filesystem abstraction provides mechanisms for creating and maintaining namespaces, it does not define how they should be organized (separation of policy from mechanism).

Consistency models: By nature, WSNs are dynamic, concurrent systems. Thus, clients' view of the namespace and even data may be inconsistent with respect to the current actual state. For example, a client may use the namespace to determine that a particular mobile sensor is in a specific region, but discover when it actually reads data from the sensor, that it has moved out of the previously determined region. Or, stale, cached sensor readings may be sent to a client as a result of transmission interruptions. Additional inconsistencies can arise from coupling between different files. For example, a client may set a sensor range by writing to a control file, but subsequently read an out-of-range value from a reading file, due to caching or other delays.

Strong consistency models could be implemented using distributed locks and other techniques, but the nature of WSN applications generally suits weak consistency models. Sensor data is by nature unreliable, and applications usually do not rely on high-quality, consistent operation. Adopting the file system abstraction, also allows us to apply research in distributed file system consistency models such as those developed within the Coda file system project [3].

Managing streaming data: Sensors typically produce stream data representing their samples over time. Stream data is not directly supported in the Styx framework; extensions to the file system abstraction to model streaming devices may be required. We are currently working on extending Styx protocol so that streaming data can be handled more efficiently.

Supporting in-network application-specific processing: Our framework supports in-network aggregation in the following two ways. In the first approach, a user can extend the existing Styx server to incorporate the required functionality. The Styx server implementation is fairly simple and easy to extend. In the second approach, the user implements the functionality within an independent dynamic library. The Styx server then loads the dynamic library at run time as needed, and unloads them when unneeded to free up memory. The dynamic library would conform to a defined interface for plug-in modules.

For applications with relatively static requirements, such as a debugging application which needs access to sensor registers and memory, the first design choice is a better option. Also, very commonly used aggregation functions including average, min, max can be implemented within the Styx server. However, for applications that require more sophisticated in-network processing or whose functionality changes more often, the second design choice is a better option.

Tolerating network unreliability: Wireless channels are susceptible to fading and interference. Furthermore, to conserve energy, sensors are often operated with a low-duty cycle, turning off their radios for extended periods of time. This intermittent connectivity poses unique challenges to filesystem design. One partial solution is to cache relatively static sensor information in the cluster head, which can then respond to queries even when communication to the sensor is interrupted.

V. Additional Capabilities

Using a file system abstraction offers additional advantages for application developers in a sensor network. Some of these are reviewed in this section.

Ease of application development: The file system interface is well understood (both semantically and syntactically) by application developers and system programmers. This interface can be easily used by scientists and researchers who are not familiar with the intricacies and low-level details of sensor network systems.

Access control via file permissions: File systems incorporate simple but flexible access control mechanisms via file permissions. For example, the permissions on a sensor control file might grant write access to the administrator group to allow calibration, while only granting read access to normal users to allow querying the current device state.

Ease of integration: Abstracting the sensor network as a filesystem may allow tools designed in other contexts to be adaptable to use in sensor network environments. This includes, for example, development and visualization tools developed for desktops, PDAs, or even distributed systems. Applications of interest can then be ported to the proposed file system abstraction with an effort significantly lower than having to develop them from scratch.

Portability across sensor architectures and protocols: The file system abstraction using the Styx protocol can serve as a bridging layer for interoperating heterogeneous sensors as well as interactions with external devices. In this sense, it plays a role similar to that played by IP in interconnecting heterogeneous networks. Once a new device has support for file system/Styx primitives, it is able to interoperate with the remainder of the system.

VI. Examples

In this section we demonstrate the use of the file system abstraction with three examples. The examples show important sensor network functionality and highlight the capabilities of the filesystem framework.

A. Sensor Monitoring and Calibration

Monitoring the resource state of sensors is an important capability for sensor networks [7]. Moreover, sensor calibration is essential for reducing the noise in the sensor data [8]. The file system provides mechanisms to discover sensors, as well as read and write their

state, which allow the application developers to rapidly and even interactively monitor and calibrate the sensor network. For example, the following commands can be issued by a client to discover the temperature sensors in an area, read the remaining energy of one of the sensors, and then write a parameter to calibrate another.

```
mount /dev/network /network
ls /network/cluster1/sensors/
cat /network/cluster1/s1/remaining-energy
echo 2.5 > /network/cluster1/s1/control
```

Note that an application-specific namespace can provide S&R functionality similarly to the example above. We may monitor for sensors reading a temperature higher than a threshold, look for actuators near them, and then control the actuators, for example, to initiate a cooling response in those areas.

B. Data-Centric Application

The second example illustrates how the filesystem abstraction supports a data-centric operation representative of a S&R system. Effective S&R operation requires in-network processing to localize interactions and reduce the size of the data transmitted by the sensors [9]. For example, data from multiple sensors can be aggregated to reduce the overall data size transported to an observer. Or, the data may be analyzed to detect events and initiate responses close to the event location, reducing the cost of data transmission and enhancing response time.

Consider an example where the average temperature in a region (region 10) is periodically reported to a monitoring stationg. We describe the planning and execution of this task from a centralized server perspective for simplicity; however, the namespaces may be maintained, and the task planning carried out, hierarchically and in a distributed fashion by multiple servers within the sensor network.

First, the application namespaces are consulted to discover the sensors in that region by using `ls /network/location/region-10/*`. This determines that cluster 1 is within the area of interest. The location information in the namespace is now used to find a set of sensors with the appropriate coverage. In addition, we may consult an energy-based namespace where sensors are categorized in terms of their remaining energy. This allows the application to avoid selecting sensors with low available energy (e.g., S_4 and S_5) leaving only high remaining energy sensors who satisfy the coverage requirements (S_1, S_3, S_6, S_7).

Resource namespaces can also support detailed query planning by tracking network-level resources—in our case to determine the routing and aggregating nodes in the network. These namespaces may include sensor connectivity, bandwidth availability, and resource availability. At the end of this step, the task planning is accomplished, and a suitable set of sensors, the dataflow in the network, as well as any in-network processing is determined.

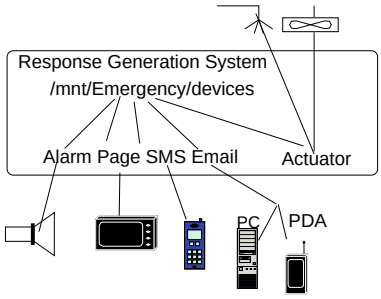

Fig. 3. An S&R system.

The query is executed as follows. The source sensors are tasked with an appropriate reporting rate (which can later be adapted) to their upstream neighbors as per the determined dataflow path. Basic sensors have support for sending and receiving packets, but some sensors (e.g., cluster heads) support Styx servers and act as multiplexers. Communication between the sensors forming the dataflow is set up using Styx. Application specific in-network processing can be accomplished by customizing packet handlers in these multiplexer nodes. This can be done dynamically (allowing specialized handlers to be moved to appropriate places in the network), statically (at compile time, or within the Styx protocol), or by allowing the application to select among a menu of predetermined handlers.

C. Heterogenous Response System Architecture

In the first example, we demonstrated how we can control actuators embedded with the sensor network to generate the required response. In this example, we describe the flexibility of the proposed framework in terms of incorporating a wide range of heterogenous devices. As an example, consider a S&R system (Figure 3) deployed in a chemical factory to detect any gas leakage. The response generation system takes input from a range of chemical sensors, processes it and then generates the necessary response. The response might include local activities such as controlling actuators embedded within the sensor network or it might include contacting external entities and authorities or in some cases a combination of both.

If the system is built using the file system abstraction, it might have a directory called /mnt/Emergency and the response generation system might organize different responses under this directory. For example, upon detecting gas leakage, it might set an alarm to alert local workers and activate the actuators on a sprinkler in order to turn it off. In addition, it might page or send SMS events to police officers and medical professionals and e-mail other local authorities.

A task force that manages crisis often consists of individuals from various government and non-government organizations. In many cases, the task force is formed in an ad hoc fashion without any knowledge about underlying sensing infrastructure [10]. With the proposed

framework a new device can be mounted on the fly under the /mnt/Emergency directory and the concerned authority can start getting the notification messages immediately. Also, inter-organization communication can be accomplished more easily using simple navigate, read, and write commands. Essentially, the filesystem abstraction operates as a unifying layer that bridges the differences in the different underlying organizational networks (much like IP does for data networks).

VII. Implementation

We have implemented a prototype which integrates the Styx protocol library with the ns-2 simulator [11]. In the current implementation, during the initialization phase, the cluster head (CH) discovers the neighboring sensors and since the CH is running the Styx file server, it simulates sensing devices as files in a file system hierarchy. In the current implementation, we have incorporated the support for constructing various namespaces within the Styx server. Then the client starts the session with the CH by calling the attach function exposed by the client-side Styx library. The client-side Styx library then encodes this command into a low-level Styx message which is sent over the wireless channel. The Styx server running on the CH interprets this incoming Styx message, processes it, and sends a pointer to its root directory to the client, again using the Styx protocol. It should be noted that, the client-side Styx library exposes a clean file system interface and hides all the low-level details of the Styx protocol from the client. Upon getting the pointer to the root directory, the client is able to navigate this directory structure using the walk command and it reads the files using the read command. In essence, the simulation set-up supports the capability required by the sensor network monitoring example described in Section VI. In addition, with our simulated prototype, we are able to simulate a sensor network consisting of at least few hundred sensors. At present, we are conducting simulations to characterize performance of the proposed file system abstraction on a large scale sensor network.

We have also developed the basic infrastructure for implementing the file system abstraction on real sensors such as the Stargate and Berkeley motes. We have developed a lightweight file server model suitable for the Motes, which consists of about 1000 lines of code and is less than 8KB in size. Our design incorporates the fact that these Motes have reasonable about of flash memory (a few KB) but much less RAM (few hundred bytes), by extensive use of static structures such as device tables and by judicious use of dynamic memory. We have also adopted the less demanding event-driven model as opposed to using runtime threads. Once this implementation is complete we hope to start using it in problems concerning resource monitoring, calibration, and distributed debugging all leading to more complex data centric applications.

VIII. Conclusion

Sense-and-respond systems are typically heterogeneous and resource-constrained. Under these conditions, system and application development is difficult, especially for domain experts and other developers whose specialty may not be embedded systems. In this paper we have demonstrated how a simple and well-known abstraction, that of a file system, hides much of the underlying complexity, allowing developers to focus on the fundamental challenges of S&R systems. Our initial results with a prototype on the ns-2 simulator suggest that such an abstraction can be practically implemented. Our next step is to port our implementation to a physical WSN such as one constructed from Stargate and Berkeley Motes.

References

[1] R. Pike, D. Presotto, S. Dorward, B. Flandrena, K. Thompson, H. Trickey, and P. Winterbottom, "Plan 9 from Bell Labs," *Computing Systems*, vol. 8, no. 3, pp. 221–254, Summer 1995. [Online]. Available: citeseer.nj.nec.com/pike90plan.html

[2] P. Juang, H. Oki, Y. Wang, M. Martonosi, L. S. Peh, and D. Rubenstein, "Energy-efficient computing for wildlife tracking: design tradeoffs and early experiences with zebranet," in *In Prof. of ASPLOS 2002*.

[3] J. J. Kistler and M. Satyanarayanan, "Disconnected operation in the coda file system," *ACM Trans. Comput. Syst.*, vol. 10, no. 1, pp. 3–25, 1992.

[4] Y. Yao and J. Gehrke, "The cougar approach to in-network query processing in sensor networks," *SIGMOD Record*, vol. 31, no. 3, 2002.

[5] S. M. Dorward, R. Pike, D. L. Presotto, D. M. Ritchie, H. W. Trickey, and P. Winterbottom, "The inferno operating system," *Bell Labs Technical Journal*, pp. 5–18, Winter 1997.

[6] R. Pike and D. M. Ritchie, "The styx architecture for distributed systems," *Bell Labs Technical Journal*, pp. 146–152, April-June 1999.

[7] Y. Zhao, R. Govindan, and D. Estrin, "Residual energy scans for monitoring wireless sensor networks," in *IEEE Wireless Communications and Networking Conference (WCNC'02)*, Mar. 2002.

[8] K. Whitehouse and D. Culler, "Calibration as parameter estimation in sensor networks," in *Workshop on Wireless Sensor Networks and Applications (WSNA) 02*, 2002.

[9] D. Estrin, R. Govindan, J. Heidemann, and S. Kumar, "Next century challenges: Scalable coordination in sensor networks," in *Proc. 5th ACM International Conference on Mobile Computing and Networking (Mobicom)*, 1999.

[10] K. M. Chandy, B. E. Aydemir, E. M. Karpilovsky, and D. M. Zimmerman, "Event webs for crisis management," in *Presented at the 2nd IASTED International Conference on Communications, Internet and Information Technology*, 2003.

[11] "Network Simulator," http://isi.edu/nsnam/ns.