A Filesystem Interface for Sensor Networks

James Horey, Jean-Charles Tournier, Patrick Widener, and Arthur B. Maccabe
Department of Computer Science
University of New Mexico
{jhorey, tournier, pmw, maccabe}@cs.unm.edu

Ann Kilzer
Department of Computer Science
Gonzaga University
akilzer@gonzaga.edu

***Abstract*—Sensor network users currently face enormous challenges, including programming difficulty and the use of unfamiliar, complex interfaces. To date, most usability research with respect to sensor networks have focused on simplifying programming by offering powerful programming abstractions. Unfortunately, such work does little to encourage ordinary users, such as application scientists, to adopt sensor networks. In order to address these issues we have developed a filesystem interface for sensor networks. By treating a sensor network as a standard Unix filesystem, users are able to use familiar tools to interface with sensor networks. This simplifies management and debugging. Similarly, programs originally designed for filesystems, such as file-sharing programs, can be used to extend sensor networks. Finally, users are also able to prototype applications using pre-existing programming environments that interface with the sensor network through the file I/O interface.**

I. Introduction

Recent success in wireless sensor network deployments [16] [13] have inspired many to envision sensor networks that will be widely used and programmed [8], by ordinary "citizen scientists" [20]. Unfortunately, this vision is hampered by many challenges users face when attempting to use this technology. Such challenges include the difficulty in programming and the lack of familiar software to analyze and view data.

Although research in simplifying programming have created advanced programming techniques, most of the technical research has not differentiated sensor network *programmers* from sensor network *users*. Programming sensor networks still requires a complicated set of software for compiling, loading, and debugging programs. The complexity of this software automatically limits users to a specialized few. Many users, including application specialists (biologists, structural engineers, etc) and citizen scientists may not be comfortable with the complexity of such tools.

Once users compile and load a program, they still face additional challenges. Programs are difficult to prototype and debug and often require an unfamiliar programming environment. Once loaded, the program is still difficult to modify, especially for users resigned to simplistic interfaces. Similarly, accessing sensor data often occurs through a specialized interface. This makes collecting, viewing, and analyzing sensor data cumbersome.

Sensor networks are also difficult to integrate with existing software, since there are no common interfaces for sensor networks from a programming point of view. The lack of standardized interfaces also makes interfacing with multiple sensor networks difficult. Ideally, users should be able to discover and integrate different sensor networks into existing software without heavily modifying the original software.

In order to address these issues, we advocate interacting with a sensor network as a virtual filesystem, similar to the Unix */proc* and */sys* filesystems. In such systems, files and directories correspond to datastructures that control the operations of a program, such as the Linux kernel. In order to explore the ramifications of using a sensor network in this fashion, we have implemented a filesystem interface for sensor networks (FISN, pronounced "fission"). FISN consists of a set of functions and datastructures residing on sensor nodes and a filesystem server that allows the sensor network to be treated as a standard, POSIX-compliant filesystem.

Once mounted, common tools, such as *ls*, *cat*, and *echo* can be used to view and update data, organize groups of sensors, and control access to data. This allows interactive use and debugging. Similarly, other software can use existing file I/O libraries to interact with the sensor network. This allows users to construct prototype sensor network applications with familiar programming environments such as *Matlab*.

FISN is currently in an early stage of development. We have implemented two prototype filesystems for TelosB motes that communicate with a filesystem server through the USB serial interface. The first filesystem exposes basic sensor data, such as thermistor values. This filesystem is used to demonstrate basic operations. The second filesystem extends the first by including a task interpreter running on the sensor nodes. Using FISN datastructures, these tasks can be constructed and debugged using common filesystem commands.

We discuss the general design of FISN in Section II and discuss implementation details in Section III along with a description of the two filesystems we have implemented. Afterwards, several different use scenarios are discussed in Section IV. We discuss related work in Section V and conclude our paper with a short discussion on future work in Section VI.

II. Architecture

FISN organizes the sensor network into a tree of files and directories. Initially, individual sensor nodes are represented as directories one step removed from a leaf of the tree, while the leaves of the tree represent data generated by the parent sensor node. As the user interacts with FISN, additional directories, representing groups of sensor nodes, and data files can be

Filesystem Command	Sensor Network Operation
ls PATH	List contents of sensor node or group
mkdir directory	Create group called directory
cp, mv filename PATH	Move data item, filename, to group/sensor identified by PATH
cat filename	Read data item from sensor node
echo string filename	Create/update data item filename with content string

TABLE I
A PARTIAL LISTING OF SUPPORTED COMMAND-LINE PROGRAMS AND THE ASSOCIATED SENSOR NETWORK OPERATION.

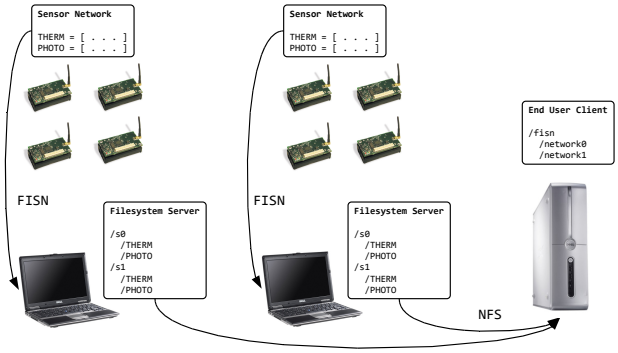

Fig. 1. The FISN architecture consists of a filesystem server that translates filesystem commands into sensor network operations.

created. Table II lists some common command line programs that can be used to interact with FISN. Not all command line programs are supported; specifically, the creation of links is currently not supported.

Each physically contiguous sensor network is controlled by a single filesystem server (Figure 1). The filesystem server is responsible for exporting the filesystem interface to the user and translating filesystem commands to sensor network operations. Additionally, the server maintains a small cache along with basic metadata, such as permissions and access control, to decrease interaction latency. Because the filesystem server is not resource constrained, the server can also export additional services, such as *NFS* or *ssh*. Ultimately, this organization allows users to interact with multiple filesystem servers over the internet.

Most data files reside on the actual sensor nodes. FISN also provides a configuration option that allows read-only files to reside on the filesystem server. These read-only files are primarily used to convey information to the user, such as the sensor platform model, and cannot be read by the programs executing on the sensor nodes. For all other data files, sensor nodes respond to commands from the local filesystem server. Information from sensor nodes are only pulled when the user explicitly requests data by initiating a filesystem command.

Instead of an interface to a static set of directories and files, FISN provides an interface for communication between the user and currently executing sensor network applications. Applications choose which data files can be viewed and controlled by the user. Likewise, the user, by using common filesystem programs, can control the operation of sensor node applications.

III. EVALUATION

In order to evaluate the filesystem semantics we have implemented two different filesystem applications for TelosB motes running Mantis OS [4]. Mantis OS provides a simple threading environment that simplifies some of our programming. However, this does not preclude the use of other operating systems, such as TinyOS [12] or SOS [11], in the future. For the filesystem server, we employ *FUSE* [1], a user-space filesystem module that simplifies the construction of Unix filesystems. Both filesystem applications we've developed use the same FUSE routines. Communication between the sensor nodes and the filesystem server is through the USB serial interface. Each sensor node has a unique device address and can be contacted directly by the filesystem server.

In order to simplify the construction of filesystems, we provide a *store* datastructure with a set of methods for retrieving and updating named data. Besides serving as a convenient way to store and retrieve data, all data in the store is automatically reflected in the filesystem. For instance, in order to expose the thermistor data as a file, the sensor node program simply reads in the thermistor data and stores the data with the appropriate name in the store. The next time the user reads the sensor node directory, the thermistor file will appear. Similarly, if a user updates or creates a file via the filesystem, the changes are automatically reflected in the store and can be examined by the sensor node program.

The store datastructure also provides a set of compile time parameters to control the number of named data, the length of data names, the number of data items associated with a name, and the maximum size of each data item. Using these parameters, developers are able to minimize the memory impact of the filesystem.

The first filesystem we developed is basic and serves sensor data, including thermistor, photometer, humidity, and the LEDs. Users are able to read sensor data and turn the LEDs on and off by writing *ON*, *OFF*, and *TOGGLE* strings to the appropriate LED file. We have also constructed a more sophisticated filesystem that includes a simple task interpreter inspired by the Tenet tasking library [7]. Like the original Tenet tasking library, we provide a set of simple functions that can be chained together to perform a particular task. At the moment, however, our task interpreter includes only a subset of the Tenet functions (Table III).

Users invoke the task interpreter by creating a file specifying the relevant functions in the intended sensor node directory. This file can be created using a normal text editor. Afterwards,

Task Command	Task Operation
sample(INPUT, OUTPUT, SAMPLES, RATE)	Collects samples from input and stores the results in output
classify(INPUT, OUTPUT, OP, THRESH)	Classifies input and store results in output
stat(INPUT, OUTPUT, OP, SAMPLES)	Performs statistics on input and store results in output

TABLE II
FUNCTIONS SUPPORTED BY OUR TASK INTERPRETER. BY WRITING THESE FUNCTIONS INTO APPROPRIATE FILES USERS ARE ABLE TO PROGRAM THE SENSOR NODE.

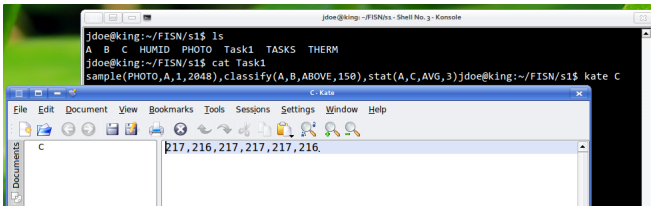

Fig. 2. Running programs may generate intermediate files that can be accessed for debugging.

the user must update the task control file with the name of the newly created task file. The task interpreter monitors the task control file for new tasks and automatically begins interpreting the task functions. Because Mantis OS supports preemptive threading, the user can specify multiple task files that are interpreted concurrently. As tasks are interpreted, tasks may create intermediate files that can also be examined via the filesystem.

The latency experienced by the user for most interactive commands is less than a second. This latency is a factor of the number of FUSE functions the command invokes and the amount of data that's transmitted from the sensor node. Even "simple" commands, such as *ls*, may invoke a single FUSE function multiple times. Also the sensor node currently transmits each data item associated with a data name in a single packet. Future work includes aggressively optimizing the code to reduce latency.

IV. APPLICATIONS

FISN can be used to interact with sensor networks in several novel ways. Besides interactive use, we envision application prototyping, debugging, and file viewing to become commonplace. Unlike previous tools that were designed for these specific purposes, FISN allows us to leverage a large body of existing software originally designed for filesystems. As importantly, FISN does not limit interaction to existing tools. As new tools for filesystems develop, they can be re-used for sensor networks as well.

Interactive Debugging

Currently, one of the difficult aspects of sensor network programming is the lack of information about currently executing programs. Once loaded onto the sensor nodes, programs are often a black box and interaction is limited. Applications that expose datastructures via FISN, however, provide a method for users to investigate the internal structure of *executing* programs. By occasionally issuing read commands

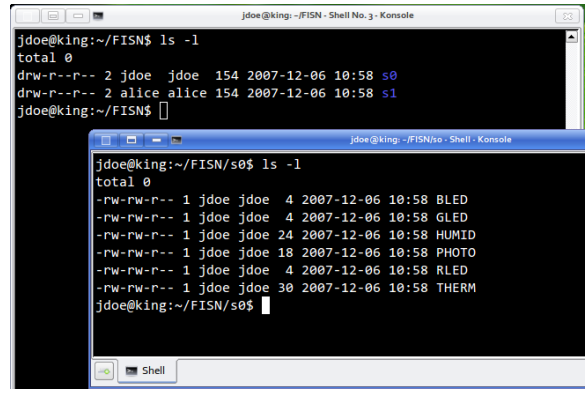

Fig. 3. Filesystem permissions and access control can be used to control access to different sensor nodes and data.

on directories and files, users can become aware of new data and probe their values. For example, when executing a set of tasks in the task interpreter, intermediate values are generated by the different tasks. These data values, like the sensor data, automatically appear as files. Figure 2 illustrates this process with a simple set of tasks that samples the photometer, records all values above a given threshold, and also calculates the running average.

Access Control

A common problem associated with end-user interfaces is access control. Typically, access control in sensor networks is coarse-grained; all users with access to the basestation can read and write data. By employing a filesystem abstraction, however, FISN is able to re-use file modes and access control mechanisms to manage the sensor network. Administrators are able to restrict access to certain sensor nodes and are able to specify read and write permission on a per-file basis (Figure 3). This scheme allows certain users and groups to create and read important system files while ensuring that everybody has access to shared sensor data. Administrators can change these modes by issuing familiar *chmod* and *chown* commands.

Application Prototyping

Many potential sensor network users may be unfamiliar and uncomfortable with existing programming environments. Such environments often require users to configure cross-compilation environments and to be familiar with complex build tools. Even in situations where users may be comfortable with such tools, the complexity of using these tools may be overkill for a particular job. In such situations, we advocate the use of FISN as an application prototyping environment.

```
function readsensors(path)
   while(true)
      sensor0_light = [sensor0_light, dlmread(strcat(path, '/s0/PHOTO') )]
      sensor0_humid = [sensor0_humid, dlmread(strcat(path, '/s0/HUMID') )]
      sensor0_temp =  [sensor0_temp, dlmread(strcat(path, '/s0/THERM') )]

      hold
      plot(sensor0_light, 'mo')
      plot(sensor0_humid, 'mx')
      plot(sensor0_temp, 'ms')
      hold off

      pause(1)
   end
end
```

Fig. 4. A Matlab script that uses the dlmread function to record and plot sensor data.

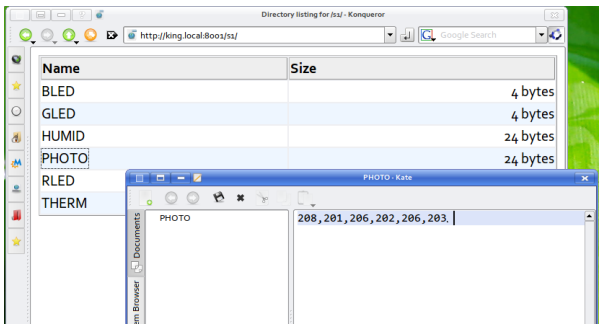

Fig. 5. By serving files via a web interface, users are able to browse sensor data using their web browser.

Figure 4 illustrates a *Matlab* script to read and plot sensor data from a single sensor node. Although simple, the script highlights how FISN can be used to integrate sensor data into existing programming environments using existing I/O functions. So long as the user's programming environment supports reading and writing files, he will able to prototype sensor network applications.

Standard Data Interfaces

Besides locally mounting the filesystem server, system administrators can also create simple methods to share data over the internet with multiple sensor networks. For instance, each filesystem server can also export an *ssh* or *NFS* server. Remote users can then mount multiple sensor networks to manage several networks using a single interface. If the user employs a secure protocol, such as *sshfs*, data exchanges from the sensor network to the remote host can be secured without much additional work. Adding an additional sensor network is also simple, since each sensor network is managed by a separate filesystem server that can be mounted remotely.

Additionally, many existing programs, such as the KDE file server applet (kpf) [2], allow users to easily configure a web server that serves public files. The kpf applet accepts a parent directory, generates a webpage containing the subdirectories and files, and announces the service over the local network using the zeroconf protocol [10]. Users on the local network will automatically be notified of a new webserver and subsequently be able to browse the contents of a sensor node (Figure 5).

By exposing the sensor network as a standard filesystem, applets, such as kpf, can be re-used to serve sensor data with minimal additional work. In the future, additional applets that are designed to interact with the underlying filesystem may provide other services, such as RSS. FISN users will benefit from these future advances automatically.

V. Related Work

FISN is originally inspired by the Plan 9 operating system [18]. In Plan 9, all resources are mapped as files. This allows complex operations to be performed on all resources using a common set of tools. Although FISN does not capture all aspects with respect to Plan 9's filesystem capabilities, FISN captures the important property of allowing the user to control sensor resources using the file I/O.

We are not the first to explore using a filesystem metaphor within the context of sensor networks. Tilak [21] and Pisupati et al. [19] propose a framework for programming sensor nodes using hierarchical filesystems. In both cases a filesystem server resides on the sensor node and presents a filesystem interface to the node applications. In FISN, however, applications running on sensor nodes do not have access to a filesystem interface; the filesystem is created at the filesystem server. This simplifies and reduces the overhead of programs running on the sensor nodes.

Besides filesystems, others have suggested a relational database model to interact with sensors networks. TinyDB [14], Cougar [24], and IrisNet [6] are examples of such approaches. Although these systems simplify interaction with the sensor network, the query language interface is not as standard as the file I/O interface making integration into existing software more difficult.

Our work also resembles various macroprogramming models. In these models, programmers use abstractions that enable them to program a large set of sensor nodes without explicit communication. For example, Abstract Regions [15] and Hoods [22] provide methods that abstract neighborhood information. Other environments, such as EnviroSuite [3] provides explicit support for tracking applications. Since these systems provide support at the node and network level, they provide complementary services to FISN.

Other macroprogramming environments, such as Kairos [9], Regiment [17] and the Declarative Sensor Network platform [5] offers a single program view of the sensor network. Programs written in such environments resemble FISN programs executing on the basestation, since we also provide a global view of the network. However, FISN is not a programming language environment, and does not provide a method to compile global level programs into node level code.

FISN also shares many things in common with recent work on interactive debugging systems. Marionette [23] provides interactive debugging support for TinyOS programs. Users are able to probe for data values and call functions on the sensor node. Besides debugging, this system can also be used to

prototype applications, by transporting values from the sensor node to the basestation which ultimately executes the logic.

VI. Conclusion and Future Work

This paper describes FISN, a system that presents a sensor network as a standard Unix filesystem. Sensor nodes are displayed as directories containing user-specified data files. We've developed two filesystems running on the sensor nodes; one that exposes basic sensor data and another more sophisticated example that implements a simple task interpreter. Like typical filesystems, users can employ existing tools, such as *ls* and *cat* to navigate, read, and control the sensor network. Also, by employing a filesystem interface, applications originally targetted to serve files, such as *NFS* and file-serving programs, can be re-used for sensor networks. Ultimately, this interface simplifies and extends the way users can interact with sensor networks. We envision a future where users are no longer encumbered by complex programming software and can readily use sensor networks in novel ways.

For the future, we plan on simplifying and supplementing the FUSE API to include additional functions, such as pre and post processing functions. This can be used, for example, to support automatically renaming a file before sending it to the sensor node. Since sensor nodes have limited memory, precaution must be taken to reduce memory consumption. By including a standardized way to support pre and post processing functions, users will be able to use long filenames without consuming additional memory on the sensor nodes.

We also plan on creating a FUSE-like API for the sensor nodes. Although we currently provide a store mechanism that allows sensor node programs to interact with the filesystem data, the filesystem programmer is still exposed to unnecessary detail regarding how messages are retrieved and formatted. Also, including additional functionality, such as the ability to fetch data from something other than the provided data store is currently unsupported. By providing a standardized node API, additional filesystems should be simpler to construct and more maintainable.

Finally, we intend on extending FISN to operate over wireless communication. This poses additional difficulties, since both energy consumption and latency is increased. In order to mitigate these effects, we are exploring statistical caching mechanisms that attempt to minimize the interaction latency while ensuring that the power and memory is not prematurely depleted.

VII. Acknowledgments

The authors gratefully acknowledge the support of the National Nuclear Security Agency (grant DE-FG52-06NA27494/A000), and the input of members in the Scalable Systems Lab at UNM.

References

[1] Filesystem in userspace: http://fuse.sourceforge.net/.

[2] Kde public file server applet: http://docs.kde.org/userguide/networking.html.

[3] T. Abdelzaher, B. Blum, Q. Cao, Y. Chen, D. Evans, J. George, S. George, L. Gu, T. He, S. Krishnamurthy, L. Luo, S. Son, J. Stankovic, R. Stoleru, and A. Wood. Envirotrack: Towards an environmental computing paradigm for distributed sensor networks. In *International Conference on Distributed Computing Systems (ICDCS)*, 2004.

[4] H. Abrach, S. Bhatti, J. Carlson, H. Dai, J. Rose, A. Sheth, B. Shucker, and R. Han. Mantis: System support for multimodal networks of in-situ sensors. In *Workshop on Wireless Sensor Networks and Applications (WSNA)*, 2003.

[5] D. Chu, L. Popa, A. Tavakoli, J. Hellerstein, P. Levis, S. Shenker, and I. Stoica. The design and implementation of a declarative sensor network system. In *ACM Conference on Embedded Networked Sensor Systems (SenSys)*, 2007.

[6] P. B. Gibbons, B. Karp, Y. Ke, S. Nath, and S. Seshan. Irisnet: An architecture for a world-wide sensor web. *IEEE Pervasive Computing*, 2(4), 2003.

[7] O. Gnawali, B. Greenstein, K.-Y. Jang, A. Joki, J. Paek, M. Vieira, D. Estrin, R. Govindan, and E. Kohler. The tenet architecture for tiered sensor networks. In *ACM Conference on Embedded Networked Sensor Systems (SenSys)*, 2006.

[8] R. Govindan. The quest for a general-purpose sensing system. In *Keynote: Workshop on Embedded Networked Sensors (EmNets)*, 2007.

[9] R. Gummadi, O. Gnawali, and R. Govindan. Macro-programming wireless sensor networks using kairos. In *International Conference on Distributed Computing in Sensor Systems (DCOSS)*, 2005.

[10] E. Guttman. Autoconfiguration for ip networking: Enabling local communication. *IEEE Internet Computing*, June 2001.

[11] C.-C. Han, R. Kumar, R. Shea, E. Kohler, and M. Srivastava. Sos: A dynamic operating system for sensor nodes. In *International Conference on Mobile Systems, Applications, and Services (MobiSys)*, 2005.

[12] J. Hill, R. Szewczyk, A. Woo, S. Hollar, D. E. Culler, and K. S. J. Pister. System Architecture Directions for Networked Sensors. In *Architectural Support for Programming Languages and Operating Systems (ASPLOS)*, 2000.

[13] P. Juang, H. Oki, Y. Wang, M. Martonosi, L.-S. Peh, and D. Rubenstein. Energy-Efficient Computing for Wildlife Tracking: Design Tradeoffs and Early Experience with ZebraNet. In *Architectural Support for Programming Languages and Operating Systems (ASPLOS)*, 2002.

[14] S. R. Madden, M. J. Franklin, J. M. Hellerstein, and W. Hong. Tinydb: an acquisitional query processing system for sensor networks. *ACM Transaction Database Systems*, pages 122–173, 2005.

[15] G. Mainland and M. Welsh. Programming sensor networks using abstract regions. In *Symposium on Networked Systems Design and Implementation (NSDI)*, 2004.

[16] A. Mainwaring, J. Polastre, R. Szewczyk, D. Culler, and J. Anderson. Wireless Sensor Networks for Habitat Monitoring. In *Workshop on Wireless Sensor Networks and Applications (WSNA)*, 2002.

[17] R. Newton, G. Morrisett, and M. Welsh. The regiment macroprogramming system. In *Information Processing in Sensor Networks (IPSN)*, 2007.

[18] R. Pike, D. Presotto, S. Dorward, B. Flandrena, K. Thompson, H. Trickey, and P. Winterbottom. Plan 9 from Bell Labs. *Computing Systems*, 8(3):221–254, Summer 1995.

[19] B. Pisupati and G. Brown. Poster: File system framework for organizing sensor networks. In *Symposium on Applied Computing (SAC)*, 2006.

[20] S. Reddy, G. Chen, B. Fulkerson, S. J. Kim, U. Park, N. Yau, J. Cho, M. Hansen, and J. Heidemann. Sensor-internet share and search: Enabling collaboration of citizen scientists. In *Workshop for Data Sharing and Interoperability - IPSN*, 2007.

[21] S. Tilak, B. Pisupati, K. Chiu, G. Brown, and N. Abu-Ghazaleh. A file system abstraction for sense and respond systems. In *Workshop on End-to-End, Sense-and-respond Systems, Applications, and Services (EESR)*, 2005.

[22] K. Whitehouse, C. Sharp, E. Brewer, and D. Culler. Hood: a neighborhood abstraction for sensor networks. In *International Conference on Mobile Systems, Applications, and Services (MobiSys)*, 2004.

[23] K. Whitehouse, G. Tolle, J. taneja, C. Sharp, S. Kim, J. Jeong, J. Hui, P. Dutta, and D. Culler. Marionette: Using rpc for interactive development and debugging of wireless embedded networks. In *Information Processing in Sensor Networks (IPSN)*, 2006.

[24] Y. Yao and J. Gehrke. The Cougar Approach to In-Network Query Processing in Sensor Networks. In *ACM SIGMOD Conference*, 2002.