Tables: A Table-Based Language Environment for Sensor Networks

James Horey, Eric Nelson, and Arthur B. Maccabe
Department of Computer Science
University of New Mexico
{jhorey, nelson, maccabe}@cs.unm.edu

***Abstract*—Intuitive programming environments targetting application specialists and casual users are currently lacking. Current work has either required large investment from users to learn advanced programming techniques or has focused on simplistic and limited tools. This paper introduces Tables, a graphical programming environment that consists of a spreadsheet-inspired interface and a local runtime executing on sensor nodes. Tables emphasizes ease-of-use by reusing spreadsheet abstractions, such as pivot tables and functions, to interactively program the sensor network. By using these familiar tools, users are able to construct complex applications that include local data filtering and collective processing.**

We discuss the design and prototype implementation of Tables and demonstrate how to use Tables to create simple environmental monitoring applications and an advanced object-tracking application. We evaluate these applications in terms of ease-of-use and the relative network overhead, measured in simulation, imposed by the Tables environment. Using this evaluation, we show that the Tables programming environment represents a feasible alternative to existing programming systems.

I. Introduction

Wireless sensor networks are becoming an important tool for a wide variety of applications including environmental monitoring [19], urban sensing [24], personal networks [23], and wildlife tracking [14]. Besides application specific uses, sensor networks are also envisioned as a general-purpose tool [8] that will enable large-scale collaboration and data analysis by both application specialists and ordinary citizen scientists [22]. However the growth of sensor network technology is hampered by the fact that sensor networks remain difficult to program and maintain. This difficulty stems from the combination of inadequate programming constructs, the distributed nature of sensor network applications, and the severe resource constraints imposed by the hardware.

Previous attempts at simplifying programming have not sufficiently considered casual users and application specialists. Although advanced programming constructs, such as functional and spatial abstractions, help programmers construct more elegant code, these constructs may be difficult to grasp by non-expert programmers. Application specialists and casual users, as a consequence, are limited to tools for simple data viewing and parameter manipulation. In order to create advanced applications, these users must interact with other more experienced programmers. This is not a long-term solution since users may require unanticipated functionality after the initial deployment.

Our goal is to create a programming environment that can be readily used by application specialists and casual users. This programming environment must be both *simple* to use and *flexible* enough to create complex applications. More specifically, any end-user oriented programming environment must achieve the following set of goals:

- Allow application specialists and casual users to easily create simple programs.
- Allow advanced users to create complex applications using the same set of constructs.
- Minimize the difficulty in learning the environment by re-using familiar concepts and interfaces.

We present Tables, a table-based language environment for sensor networks that achieves these goals. In Tables, users view the sensor network as an interactive, distributed spreadsheet. Using graphical data organization tools and functions users can iteratively create simple and complex applications.

In this paper, we discuss the design and prototype implementation of Tables (Section II). This discussion is illustrated by two environmental monitoring applications. We also demonstrate how to employ advanced Tables concepts to create a complex application object tracking (Section III). Afterwards, we discuss implementation challenges and using a sensor network simulation, we evaluate Tables and demonstrate that it has modest network traffic overhead (Section IV). We show that this overhead is largely an artifact of the interactive style of programming. Our paper also identifies and discusses key system challenges that may be shared with other, future interactive programming systems (Section VI). Finally, we offer concluding remarks in Section VII.

II. Tables Concepts

In order to minimize the number of new concepts needed to program a sensor network, we designed and implemented a programming environment inspired by the spreadsheet. Spreadsheets are familiar to many computer users and include advanced data manipulation functionality. In a typical spreadsheet environment, such as Microsoft Excel, users are presented with data placed along three axes: rows, columns, and sheets. The data can then be manipulated by functions that operate over a range of cells. The output of the function can then, in turn, be consumed by additional functions. More recently, advanced spreadsheet applications also include

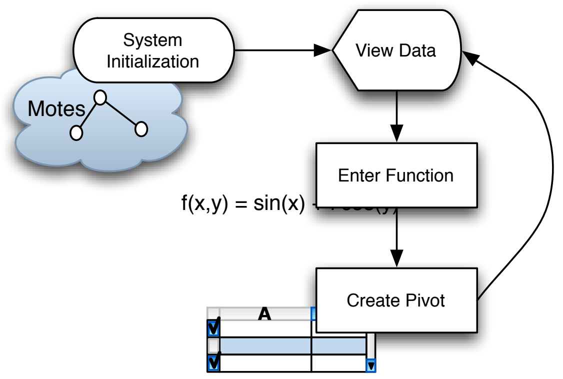


Fig. 1. The Tables workflow: users interact with Tables by iteratively viewing data, specifying functions to filter that data, and viewing data again.

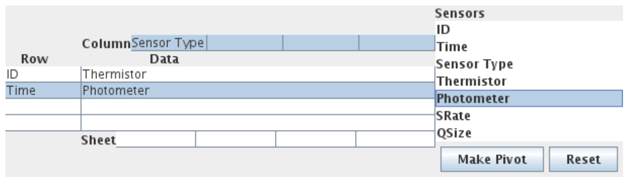


Fig. 2. A pivot table where the user is requesting to view the thermistor and photometer data organized by the node ID, time, and the sensor type. The user clicks and drags the appropriate item to one of the four panes to organize the spreadsheet.

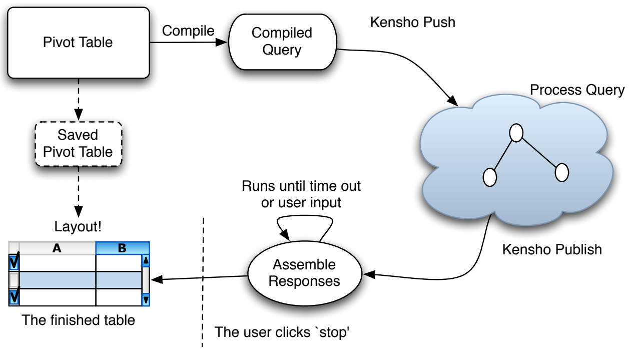
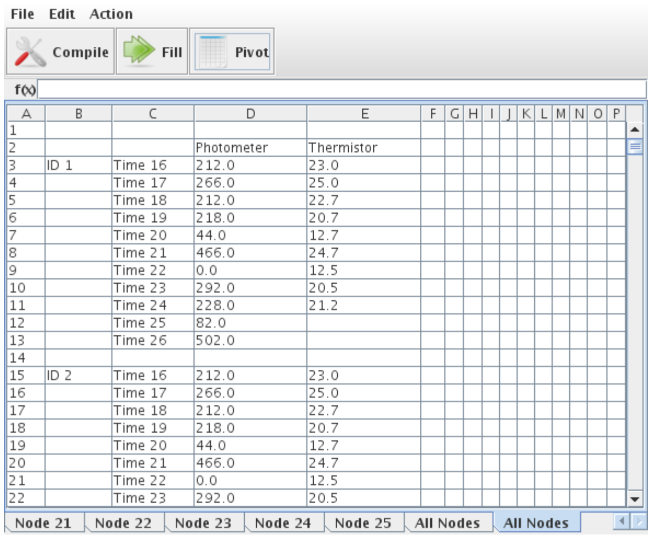


Fig. 3. The pivot table is compiled and propagated onto the sensor network. Each sensor node executes a query processor that accepts pivot tables and forms responses.



	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P
1																
2				Photometer	Thermistor											
3		ID 1	Time 16	212.0	23.0											
4			Time 17	266.0	25.0											
5			Time 18	212.0	22.7											
6			Time 19	218.0	20.7											
7			Time 20	44.0	12.7											
8			Time 21	466.0	24.7											
9			Time 22	0.0	12.5											
10			Time 23	292.0	20.5											
11			Time 24	228.0	21.2											
12			Time 25	82.0												
13			Time 26	502.0												
14																
15		ID 2	Time 16	212.0	23.0											
16			Time 17	266.0	25.0											
17			Time 18	212.0	22.7											
18			Time 19	218.0	20.7											
19			Time 20	44.0	12.7											
20			Time 21	466.0	24.7											
21			Time 22	0.0	12.5											
22			Time 23	292.0	20.5											

Fig. 4. Results of a pivot table requesting thermistor and photometer data organized by node ID, time, and sensor type.

functionality (pivot tables) to automatically organize data according to user specified parameters.

Tables, by adopting the spreadsheet metaphor, emphasizes an interactive, iterative method of programming. Data is collected from the sensor network and organized using a graphical tool called the pivot table. Once the user views the data, he has the option of inputting functions that operate over that data. This function, in turn, is propagated to the sensor network to either generate new data or filter existing data. Finally, as Figure 1 illustrates, the user can then create a new pivot table to view the updated data. This workflow encourages users to treat data viewing as an integral part of the programming process.

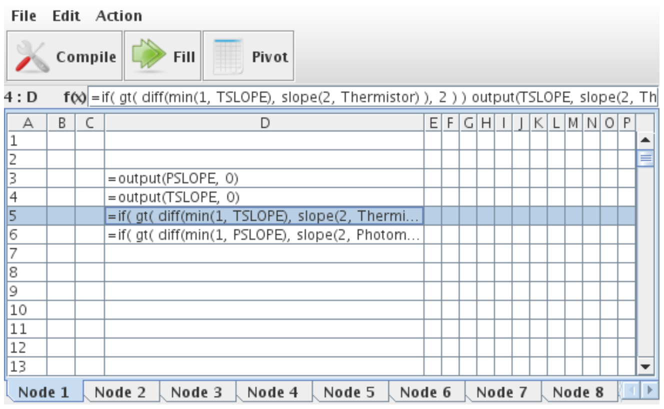
In the following sections we construct two environmental monitoring applications that demonstrate the use of various Tables components in detail. Environmental monitoring applications are straightforward and constitute the largest number of sensor network applications. Deployments include Great Duck Island [3], volcano monitoring in Ecuador [26], the Redwood forest in California [25], and bridge monitoring [15]. In this class of application, the user is interested in collecting and viewing both processed and unprocessed data from the sensor network. The data may be subsequently stored in an external database for future analysis.

A. *Pivot Tables*

In Tables, the user specifies which data he is interested in viewing by using a tool called the pivot table. The pivot table provides a miniature representation of the spreadsheet in which users click and drag data items to one of several data panes. There is a pane for the actual data and three other metadata panes, one for each of the standard spreadsheet axes (row, column, and sheet). The metadata panes specify how the items in the data pane are to be organized in the spreadsheet. Each of these panes, with the exception of the sheet pane, can contain multiple items.

The pivot table list initially contains items for the sensor values and sensor metadata such as node ID and available sensor types. As the user interacts with Tables and specifies new data elements using functions, this list is automatically updated with the new data elements. This allows the user to use the pivot table to iteratively retrieve both built-in data along with user generated data.

The pivot table can be used to create a simple environmental monitoring application. Figure 2 illustrates a simple pivot table where the user requests thermistor and photometer data organized by node ID, time, and the sensor type. Once the user specifies a pivot table, the pivot table is compiled



4 : D f(x) =if(gt(diff(min(1, TSLOPE), slope(2, Thermistor)), 2)) output(TSLOPE, slope(2, Th

A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P
1															
2															
3			=output(PSLOPE, 0)												
4			=output(TSLOPE, 0)												
5			=if(gt(diff(min(1, TSLOPE), slope(2, Thermi...												
6			=if(gt(diff(min(1, PSLOPE), slope(2, Photom...												
7															
8															
9															
10															
11															
12															
13															

Fig. 5. A set of functions that perform local filtering on the sensor node.

```
Local Functions:
      (1)
      =PSLOPE := 0
      =TSLOPE := 0
      =if( |TSLOPE - slope(2, Thermistor)| > 2) TSLOPE := slope(2, Thermistor)
      =if( |PSLOPE - slope(2, Photometer)| > 2) PSLOPE := slope(2, Photometer)

Collective Functions: (None)
Pivot Tables:
      (2) Data = PSLOPE, TSLOPE : metadata = Node ID, Time, Sensor Type
```

Fig. 6. A summary of the filtered monitoring application. The user first specifies basic filtering functions that record changes in the slope. Subsequently, the user specifies a pivot table to retrieve the slope information.

and propagated onto the sensor network (Figure 3). In Tables, each sensor node is initialized with a sampling routine that continuously collects and stores environmental data in a circular buffer. Each sensor node also executes a query processor, which processes pivot tables and forms responses. The response consists of the entire queue of data along with the specified metadata. Because the queue may be large, the response is often split up into multiple packets. After assembling all the responses from the sensor network, Tables organizes the responses to a final view according to the original pivot table specification (Figure 4).

Unlike many previous environmental monitoring applications, Tables does not employ a continuous data collection model. Instead, users are expected to construct a new pivot table each time he wants to collect data from the sensor network. Users are, however, able to specify both the sampling rate and queue size of the collected data.

B. Local Functions

Although it is often desirable to view unprocessed sensor data, filtering or compressing data may save power with only a small loss in resolution. In order to do this, Tables provides various functions that allow users to operate over sensor data and to conditionally output new data. These functions are typed into empty cells of the spreadsheet and are automatically evaluated whenever data that the function requires is updated. This, in turn, may further generate other data values which triggers the evaluation of other functions. Unlike the traditional

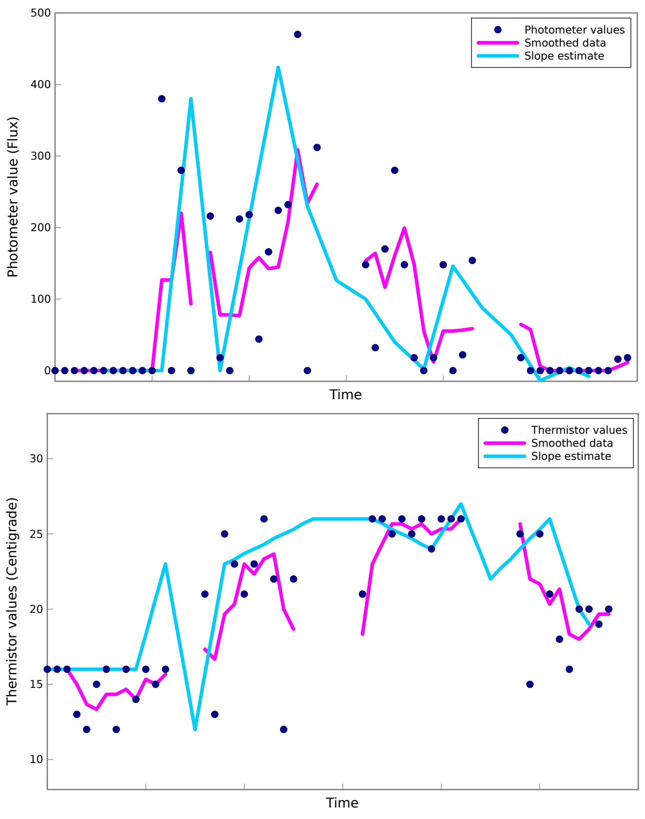


Fig. 7. Results of using Tables to collect thermistor and photometer data along with changes in the slope.

spreadsheet, however, Tables functions are executed on the sensor network.

To operate over data, Tables provides arithmetic functions, such as addition and multiplication, and vector functions, such as summation and slope. Vector functions take both a window size and the name of the data to operate over. For instance, users can specify that the *sum* function operate over the last three thermistor values. Tables also provides conditional and boolean functions that allow users to take different actions depending on the results of other computation. Finally, Tables provides assignment functions that generate new data. Upon evaluating an assignment function, Tables will store the appropriate value on the sensor node and update the pivot table list with the assigned name.

Currently, the user must manually specify the name and window size for vector functions. This is not ideal, however, since most spreadsheet users are accustomed to specifying the range using syntax similar to: *A5:A10*. This requires Tables to infer both the range and the type of data being operated over. For instance, if the *A* column contained thermistor values, Tables should infer it must operate over the last 5 thermistor values. This extension is a subject of future work.

In order to demonstrate the use of local functions, we extend the environmental monitoring application by including local compression functions that generate compressed data. In our example, the functions record changes in the thermistor and

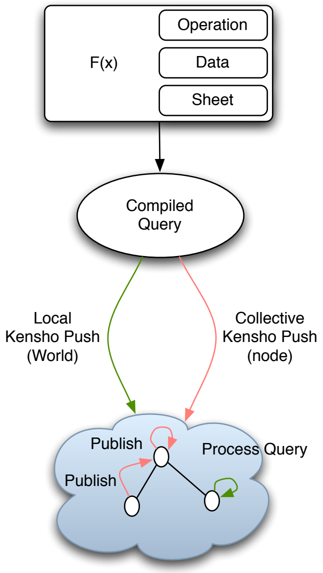


Fig. 8. Users specify functions in the Tables interface. The functions are then propagated to the sensor nodes. Collective functions operate over data that is published by group members.

photometer slopes. The user begins by first specifying the various compression functions (Figure 5). These functions are then copied onto each sheet using the *Fill* command. By default, each sheet represents a single sensor node. As such, these functions are executed by every sensor node in the network.

In order to complete the slope monitoring application, the user must wait for the sensor network to collect sufficient data to generate the slope information, and finally specify a pivot table requesting the slope information. This series of actions is summarized in Figure 6. Figure 7 illustrates the results of extrapolating the slope to graph the temperature and light levels along with the results from the original unfiltered monitoring application.

C. Collective Functions and Sheet Groups

In the slope monitoring application, the user specified a *local* filtering function. Tables also provides a simple method using pivot tables to construct *collective* functions that operate over a group of sensor nodes. When a pivot table specifies an item in the sheet pane, multiple sensor nodes may contribute to the data located on a sheet. For example, by specifying the *sensor type* item in the sheet pane all nodes with a *photometer* device will contribute data to the appropriate sheet.

If the function is specified for a sheet representing multiple sensor nodes, a logical group is defined over the relevant set of nodes. This logical group is constrained by the sheet data item and value, and each sensor node periodically evaluates the sheet constraint to determine group membership. This allow sensor nodes to leave and join the logical group over time. After group formation, a leader is elected and begins executing the functions associated with the sheet.

```
Local Functions:
      (1)
      =DETECTION := 0
      =DISTANCE := 0
      =if( Magnetometer > 0 ) DETECTION := 1 else DETECTION := 0
      =if( DETECTION == 1 &&
           Magnetometer > 7 ) PUBLISH := 1 else PUBLISH := 0
      =if( PUBLISH == 1 ) WY := Magnetometer * YLoc
                          WX := Magnetometer * XLoc
                          SENSOR := Magnetometer

Collective Functions:
      (3)
      CentroidX := sum(ALL, WX) / sum(ALL, SENSOR)
      CentroidY := sum(ALL, WY) / sum(ALL, SENSOR)

Pivot Tables:
      (2) Data = Node ID : Metadata = DETECTION
```

Fig. 9. A summary of the mobile object tracking application. This application uses sheet groups to calculate the centroid of the vehicle as the vehicle moves.

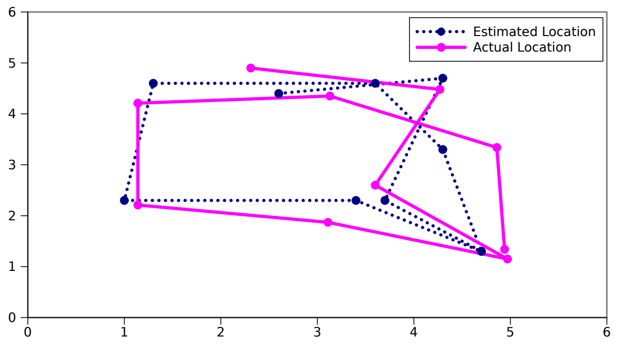


Fig. 10. Results of the tracking application. The centroid locations can be retrieved using a pivot table since user generated data appears in the pivot table item list.

Since the functions operate over a group of sensor nodes, the functions require different sensor and data values from the group members to be transmitted to the group leader. To accomplish this, the group leader sends a *publication* request to the group members, and group members subsequently *publish* the requested data values. The collective function then operates over the received values. Each group member continually transmits updated values to the group leader so that the collective function is automatically re-evaluated with new data.

Collective functions allow Tables users to construct sophisticated applications that require many-to-one communication. Unlike existing programming languages, all communication in Tables is implicit and specified by the interaction of pivot tables and functions. This simplifies network programming since users do not need to be aware of explicit message handling. In the next section, we discuss an application that uses pivot tables and collective functions to implement mobile object tracking.

III. Mobile Object Tracking

In order to demonstrate how to use pivot tables and collective functions to implement complex applications, we developed a mobile object simulator and an associated tracking application using Tables. In this application, a vehicle starts at some random position and moves in a pseudo-random manner. 25 sensor nodes are placed in 1 unit increments in a grid layout. When the vehicle is within a predefined radius of a proximity sensor, the sensor registers a positive value. The goal of this application is to employ the following centroid method to track the vehicle:

$$c_x = \sum_i R_i x_i / \sum_i R_i$$
$$c_y = \sum_i R_i y_i / \sum_i R_i$$

This application, unlike earlier examples, requires multiple pivot tables and collective functions (Figure 9). The user begins by first specifying a set of functions that determine whether a sensor node is within detection range. The first set of functions sample the proximity sensor to determine if a vehicle is within detection range. Upon detecting the vehicle, a set of local functions sets the DETECTION bit and calculates the weighted locations ($R_i x_i$ and $R_i y_i$).

After applying these functions to the sensor network, the user constructs a pivot table specifying that the DETECTION bits be placed along the sheet axis. This ensures that all sensor nodes that have recently detected the vehicle will be represented in a single sheet. After constructing this pivot table and viewing the response, the user inputs the centroid function. This function operates over the weighted location values produced by the sensor nodes. Since the function is specified on a sheet representing multiple nodes, compiling the function initiates the creation of a logical group (the sensor nodes within detection range of the vehicle). The leader of this group, in turn, requests the weighted location values from the group members and evaluates the centroid function with those values.

After allowing the application to run over a sufficient time period, the leader will accumulate several centroid locations. These locations can be retrieved by creating a final pivot table. Results from this application are illustrated in Figure 10. Compared to previous versions of this application that used a neighborhood abstraction [18], the Tables version has fewer lines of code (approximately 9 compared to 16) and employs a more interactive design. In our current implementation, we used an additional local function to further test whether the proximity value should be published by comparing the value to a threshold. Although this seems redundant (since we could easily do this in the DETECTION function), we did this to minimize the number of leader elections that result from our particular implementation. Future versions of the software will not have this issue and contain fewer lines of code.

IV. Evaluation

In this section, we discuss the implementation challenges of Tables along with the current implementation. We also evaluate the network characteristics of the previous applications. The network evaluation is used to determine the relationship between the user interactions and the network profile along with the relative overhead of this interaction.

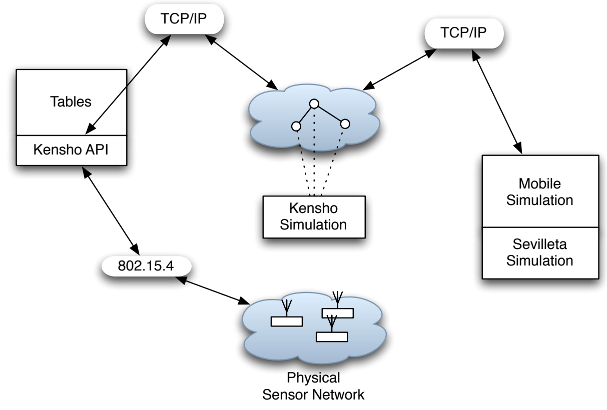


Fig. 11. The Tables interface uses the Kensho API to interact with different sensor networks. Simulated networks interact with various environmental simulators.

We also consider the relative network performance of the monitoring and tracking applications and demonstrate how by employing filtering functions and sheet groups, we can greatly reduce overall network traffic.

A. Implementation

Tables is implemented as a cross-platform Java application that interacts with the sensor network using an external communication library. As the previous applications (Section II and Section III) illustrated, Tables requires a complex messaging infrastructure. For instance, the communication library must be able to propagate pivot tables to the sensor network and receive responses from multiple sensor nodes. Also, the communication library must support collective functions that require sensor nodes to organize themselves into communication groups. Within these groups, the library must support mechanisms to allow group members and group leaders to exchange data. Although Tables does not strictly depend on any particular communication library, the library must be able to support these functions.

Currently Tables employs the Kensho tasking library [13] as its communication library. Kensho includes a C library that provides a client API that include methods to easily distribute and collect data from the sensor network along with methods to locally and collectively task sensor nodes. Specifically, the Kensho client API provides the following services:

- Pushing - The group leader is able to disseminate data to all group members.
- Publishing - Group members are able to send data to the group leader.
- Compute - Tables is able to assign both group members and group leaders with functions.

The Kensho client library connects to a simulated sensor network. Each simulated sensor node is linked with a local Kensho runtime library. This simulated sensor network, in turn, connects to different environmental simulators (Figure 11). We are actively porting the Kensho runtime to existing sensor network platforms. Because Tables interacts directly with the Kensho API, this should allow Tables to interact with physical sensor networks in the future.

Finally, Tables requires the propagation and execution of functions on sensor nodes. Currently, this is done using a function interpreter executing on each sensor node. Functions are compiled into compressed strings, propagated to the appropriate sensors, and interpreted by the sensor node. This approach is the most simple and can be employed in most operating system environments. However, virtual machine environments[16] or operating systems that feature dynamic binary linking[11] can also be used.

B. Experimental Setup

All our network evaluation is performed in simulation. Twenty-five sensor nodes are placed in a 5 by 5 grid for both environmental monitoring applications and the tracking application. Although the evaluation could have simulated more sensor nodes, this would have made it more difficult to view the overall network profile and not aided in understanding the relationship between the user actions and the network traffic.

In our simulator, sensor nodes, upon system initialization, creates a spanning tree for routing purposes. Within a single application, the routing tree does not change. We employ a network model that assumes that the probability of dropping a packet transmitted between two nodes is inversely proportional to their distance. This probability is perturbed with noise drawn from a normal distribution.

Kensho *push* commands are sent from the group leader to all members of the routing tree. Likewise, Kensho *publish* commands results in messages being routed up the tree without data-aggregation. However, if a sensor node attempts to publish multiple data values from a single data source (such as the photometer), the sensor node will compress the data as much as possible (approximately four data values per message).

C. Network Profile

Figure 12 illustrates the relationship between user interactions (creating a pivot table, specifying functions, etc.) and the network profile for all three applications. In this figure, each point represents a sensor node that has transmitted a packet. Over time, a single sensor node may participate in multiple message transmissions, although the identity of individual sensor nodes is not included in the figure.

For all applications, the system starts with group initialization, which forces all sensor nodes to organize along a spanning tree, and function assignment. All nodes are tasked with basic functions such as: the environmental sampling routine, the query processor, and the Tables interpreter. For the object tracking application, additional functions are assigned to properly handle the publication of data values. Overall we observe that these initialization messages constitute a small proportion of the overall network traffic on an individual node basis.

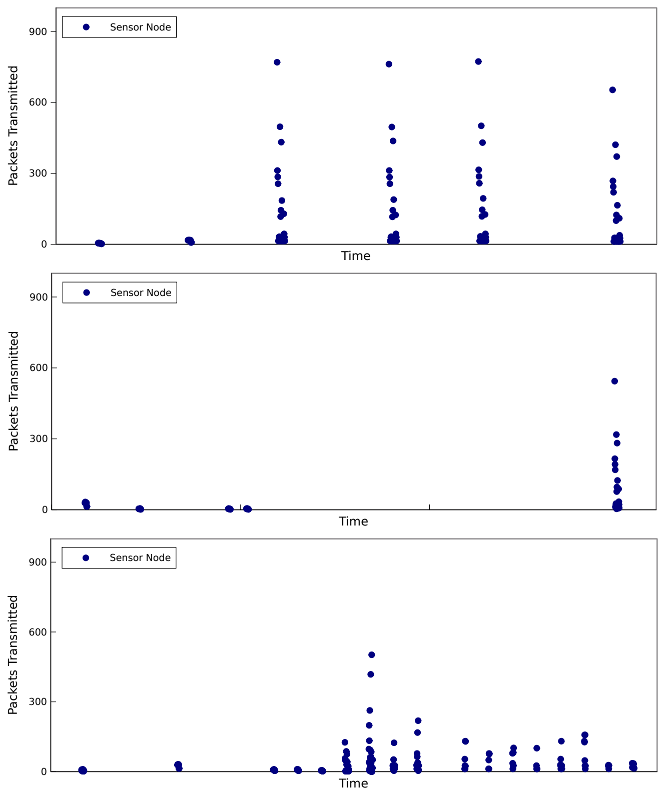


Fig. 12. Each sensor node, represented by a data point, transmits a certain number of messages in a given time window.

For the environmental monitoring application, each spike in network traffic correlates with a pivot table action. We clearly observe four pivot tables being propagated and returned. For the filtered monitoring application, additional network traffic (beyond the group initialization and function assignments) is generated by the local filtering functions. Afterwards, there is no network activity until the pivot table retrieving the slope data.

We also observe that for both monitoring applications, some nodes transmit an order of magnitude more packets than others. This is primarily due to the routing tree structure, since the root node of the routing tree must forward all the children's' packets.

Unlike the monitoring applications, the tracking application profile differs by including network traffic generated not by the user, but by the movement of the vehicle. Each time the vehicle moves, new nodes must join the sheet group and request publication data. Afterwards, all group members regularly publish location estimates to the leader. Although there are frequent network traffic, we can observe that the leader processes fewer packets compared to the monitoring applications. This is primarily because few sensor nodes actually detect the vehicle at any given time.

We draw two conclusions from this data. First, that for applications that primarily employ pivot tables, a single routing tree structure is insufficient. Alternative routing mechanisms by which the network load is shared more evenly across the

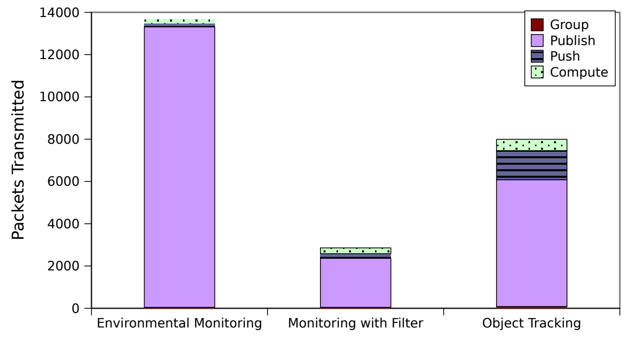


Fig. 13. Comparison of total network traffic of the different applications.

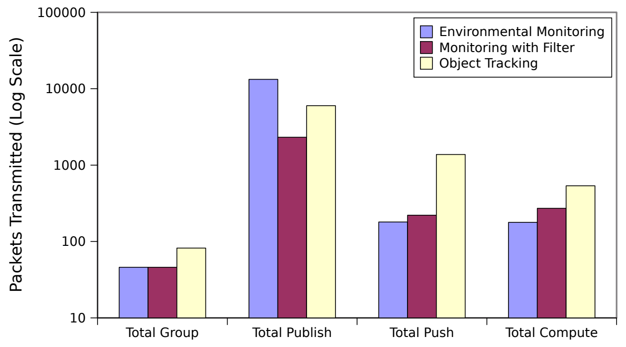


Fig. 14. Overall network traffic data with respect to the different Kensho action for both monitoring and tracking applications.

network are necessary to ensure that all nodes deplete their power uniformly. Second, the network profile is very useful when attempting to understand the overall network behavior of the system. Although it may increase communication cost, presenting this information in tandem with the different Tables actions may be useful for users and should be integrated into future designs.

D. Total Traffic

As Figure 13 illustrates, both local and collective functions affect the total network traffic in Tables. In this graph, the total number of packets transmitted by each sensor node is summed for each of the applications. The environmental monitoring application with no filtering, consumes the largest amount of network traffic. This traffic is dominated by the *publish* cost generated by the pivot table responses. This is expected since in this application, the user simply requests data at regular intervals. Also, since the sensor network does not employ data aggregation at the subtrees, all messages are relayed through the parent nodes causing a large number of packets to be relayed.

For the monitoring application with filtering, we observe that although the filtering functions are transmitted to the network, overall traffic is greatly reduced. The filtering mechanism allows the user to employ a single pivot table to reconstruct the data.

The tracking application, like the filtered monitoring application, consumes less network traffic than the environmental monitoring application, even though it transmits the centroid functions and regularly transmits publication data. This is because few nodes participate in tracking the vehicle at any given time. Also, unlike the monitoring applications, only the group leader responds to the sole pivot table requesting estimated location data.

Currently Tables uses two different methods to reduce overall network traffic. First, the user can construct an application that filters the amount of data each node collects. This allows the user to reconstruct the entire dataset with fewer pivot tables. Also, the user can construct an application whereby fewer nodes participate in the data generation and the pivot table. Another method to reduce network traffic involves aggregating or compressing data along the routing path. Tables currently does not do this, although extensions to the pivot table are being considered for this purpose.

E. Relative Overhead

Compared to non-interactive programming environments, Tables must regularly transmit queries and functions in order to operate. To quantify this relative overhead, we compare the amount of traffic generated for each of the major Kensho operations for both the monitoring and tracking applications (Figure 14). Operations that involve group initialization (*Group*), function assignment (*Compute*), and the transmission of pivot tables and functions (*Push*) are considered relative overhead. In a traditional, static programming environment, these operations would be performed before deployment.

For environmental monitoring, the *publish* costs dominate the overall network cost. This implies that the group initialization, function assignment, and function propagation play a small role and constitute relatively little overhead. For filtered monitoring, function assignment and propagation play a larger role. However, the cost is still dominated by the pivot table results. In our examples we employed the minimum number of pivot tables to reconstruct the data. However, by employing multiple pivot tables over a longer period of time we expect the relative overhead to decrease.

Finally, for the tracking application, we observe a higher relative overhead. This is because the tracking application contains more functions. Also, sheet group leaders must transmit the necessary publication data to new group members. Since sensor nodes leave and join the sheet group regularly, the relative overhead for this operation is higher. We can potentially reduce this overhead by allowing nearby group members to provide the necessary publication data to new group members. This ensures that the request message does not propagate all the way to the leader and back. This optimization is a subject of future work.

V. Related Work

Previous work in sensor network programming and management can be broadly classified into two areas: programming languages and graphical environments. In the area of programming languages, most previous work has concentrated on developing language constructs or libraries to simplify sensor network programming. However, these works often assume that users are already familiar with basic sensor network programming and as a consequence, may not be ideal for casual users and many application scientists. Previous graphical environments have attempted to simplify sensor network data viewing and management, but have been too inflexible for complex use.

A. Programming Languages and Environments

Typically sensor network applications are developed using a low-level programming language, such as NesC [6] or C. NesC is tightly integrated with TinyOS [12] and emphasizes the use of reconfigurable components with an event-driven split-code model. NesC is primarily designed for system developers that are able to leverage their knowledge of the system to create programs with minimal overhead. However, this design makes NesC a challenging programming environment even for experienced programmers.

Due to the difficulty of programming individual sensor nodes, various macroprogramming models have been proposed. In these models, programmers are given abstractions that enable them to program a large set of sensor nodes simultaneously. These sensor nodes may be organized by a logical grouping, geographic proximity, or hierarchically. Such models simplify sensor network programming by abstracting explicit communication between sensor nodes.

Examples of macroprogramming languages that abstract neighborhood information include Abstract Regions [18] and Hoods [27]. EnviroSuite [2] offers special support for tracking applications and Kairos [10] offers a single program view of the sensor network. Regiment [21], offers a functional macroprogramming language and environment. Macroprogramming systems that emphasize the role of tasking in a sensor network include Tenet [9], which offers an environment in which operations are tasked between a basestation and sensor nodes according to the computational complexity.

Other systems have attempted to lower the barrier of entry by employing a database-inspired environment where users employ a querying language, such as SQL [5] or XML. Examples of such systems include TinyDB [17], Cougar [29], and IrisNet [7]. Although these systems allow non-expert users to interact with the sensor network, the query languages employed often lack expressive power. This makes implementing complex applications, such as object tracking, very difficult and may require implementing application-specific logic using a different set of language constructs.

B. Graphical Environments

Viptos [4] and TOSDev [20] are examples of integrated development environments for NesC. These environments do not change the basic programming model offered by NesC and still suffer from the same negative aspects. Remembering and accessing NesC events becomes easier, but the components are still assembled in the same manner.

Crossbow's Mote View [1] offers a simple interface that allows users to view sensor data, routing information, and basic sensor node statistics. The interface can also be used to reconfigure application parameters, such as sampling rate. However, programming complex tasks are not possible in this environment.

Similarly, Woo et. al. presents a spreadsheet environment for managing and viewing data [28]. Unlike our work, their environment employs XML schemas to define the relationship between the sensor network and spreadsheet. At the moment, their environment performs the computation centrally on the spreadsheet and does not provide a mechanism to program the nodes directly.

VI. Future Work

In this section, we review future optimizations and enhancements to the Tables interface. Many of these optimizations and enhancements result from lessons learned while implementing the monitoring and tracking applications using Tables and identifying key difficulties and deficiencies.

As demonstrated by the previous applications, Tables emphasizes applications wherein sensor nodes, organized into groups, perform computation centrally on independently published data. However, applications that require neighborhood information may be difficult to implement. We are currently exploring the possibility of *cross-sheet* references in order to facilitate neighbor communication. Functions would refer to cells located in neighboring sheets under the assumption that sheets represent geographic neighbors. Key user-interface challenges include compressing the neighborhood information along a one-dimensional line (the sheet axis) and translating node-centric data (a node's local neighbors) to a global layout (all nodes and their neighbors).

Another key challenge in Tables is synchronization between sensor nodes in a sheet group. Nodes in a group must synchronize events such as data publication and function execution. For example, since a function on a group leader may wait for multiple data values to be published before executing, it is important that all group members publish their data at approximately the same time. Otherwise, data that is published outside the time window may be discarded or cause the premature execution of a function.

Similarly, because all functions run periodically, sensor nodes in a sheet group should synchronize the execution of these functions. This allows the collective function to operate under the assumption that published data is measuring the same phenomena in the same time window. We plan on investigating these synchronization issues by integrating barrier semantics into the underlying middleware system used by Tables.

Another major challenge is user interaction latency. Currently, the time between Tables actions and responses can be high. For example, a user creating a pivot table must wait for the query to be propagated to the network, for the response to be constructed, and to have the response sent back. Since the goal of Tables is to emphasize an interactive model for sensor network programming, this high latency may frustrate users. We are currently investigating the use of caches to minimize this latency at the possible expense of increased communication.

VII. Conclusion

In this paper, we introduced an interactive programming environment called Tables. We demonstrated that unlike previous programming languages, Tables is designed for application scientists and casual users to create both simple and complex applications. This is achieved through an intuitive spreadsheet inspired environment. Users are able to construct pivot tables to view data and to use local and collective functions to program the sensor network. Communication and group formation are handled automatically by the system to preserve spreadsheet semantics.

We demonstrated how to use Tables to construct several applications including environmental monitoring, monitoring with simple data compression, and mobile object tracking. These applications were constructed using standard Tables actions. We demonstrated using a network simulation, that relative network overhead is low for many applications and that much of the network overhead is due to the interactive design of the system. Finally, we identified and discussed key system and interface challenges.

We are exploring expanding the Tables interface to include additional features and enhancements, while ensuring that the user interface does not become too complicated. We are also currently implementing the Tables interface to work on a physical sensor network. With these advancements, we are confident that sensor networks will transform from limited-use devices to a more widely used computational tool.

VIII. Acknowledgments

The authors gratefully acknowledge the support of Los Alamos National Laboratory (grant DE-AC52-06NA25396), the NNSA (grant DE-FG52-06NA27494/A000), and the input of members of the Scalable Systems Lab at UNM.

References

[1] Crossbow: www.xbow.com.
[2] 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 *ICDCS*, 2004.
[3] A. Cerpa, J. Elson, D. Estrin, L. Girod, M. Hamilton, and J. Zhao. Habitat monitoring: Application driver for wireless communications technology. In *ACM SIGCOMM Workshop on Data Communications in Latin America and the Caribbean*, 2001.
[4] E. Cheong, E. A. Lee, and Y. Zhao. Viptos: a graphical development and simulation environment for tinyos-based wireless sensor networks. In *SenSys*, 2005.
[5] C. J. Date. *A guide to the SQL standard*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 1986.
[6] D. Gay, P. Levis, R. von Behren, M. Welsh, E. Brewer, and D. Culler. The nesc language: A holistic approach to networked embedded systems. In *Programming Language Design and Implementation (PLDI)*, 2003.
[7] 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.
[8] R. Govindan. The quest for a general-purpose sensing system.
[9] R. Govindan, E. Kohler, D. Estrin, F. Bian, K. Chintalapudi, O. Gnawali, R. Gummadi, S. Rangwala, and T. Stathopoulous. Tenet: An architecture for tiered embedded networks. Technical report, Center for Embedded Networked Sensing, 2005.
[10] R. Gummadi, O. Gnawali, and R. Govindan. Macro-programming wireless sensor networks using kairos. In *DCOSS*, 2005.
[11] C.-C. Han, R. Kumar, R. Shea, E. Kohler, and M. Srivastava. Sos: A dynamic operating system for sensor nodes. In *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 *ASPLOS*, 2000.
[13] J. Horey, A. B. Maccabe, and A. Mielke. Kensho: A dynamic tasking architecture for sensor networks. In *Workshop for Wireless Sensor Network Architectures - IPSN*, 2007.
[14] 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 *ASPLOS*, 2002.
[15] S. Kim, S. Pakzad, D. Culler, J. Demmel, G. Fenves, S. Glaser, and M. Turon. Health monitoring of civil infrastructures using wireless sensor networks. In *IPSN*, 2007.
[16] P. Levis and D. Culler. Maté: A Tiny Virtual Machine for Sensor Networks. In *ASPLOS*, 2002.
[17] 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.
[18] G. Mainland and M. Welsh. Programming sensor networks using abstract regions. In *NSDI*, 2004.
[19] A. Mainwaring, J. Polastre, R. Szewczyk, D. Culler, and J. Anderson. Wireless Sensor Networks for Habitat Monitoring. In *WSNA*, 2002.
[20] W. P. McCartney and N. Sridhar. Tosdev: a rapid development environment for tinyos. In *SenSys*, 2006.
[21] R. Newton, G. Morrisett, and M. Welsh. The regiment macroprogramming system. In *IPSN*, 2007.
[22] 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.
[23] S. Reddy, A. Parker, J. Hymanv, J. Burke, M. Hansen, and D. Estrin. Image browsing, processing, and clustering for participatory sensing: Lessons from a dietsense prototype. June 2007.
[24] M. Srivastava, M. Hansen, J. Burke, A. Parker, S. Reddy, T. Schmid, K. Chang, G. Saurabh, M. Allman, V. Paxson, and D. Estrin. Network system challenges in selective sharing and verification for personal, social, and urban-scale sensing applications. In *HotNets*, 2006.
[25] G. Tolle, J. Polastre, R. Szewczyk, D. Culler, N. Turner, K. Tu, S. Burgess, T. Dawson, P. Buonadonna, D. Gay, and W. Hong. A macroscope in the redwoods. In *SenSys*, 2005.
[26] G. Werner-Allen, J. Johnson, M. Ruiz, J. Lees, and M. Welsh. Monitoring volcanic eruptions with a wireless sensor network. In *European Workshop on Wireless Sensor Networks*, 2005.
[27] K. Whitehouse, C. Sharp, E. Brewer, and D. Culler. Hood: a neighborhood abstraction for sensor networks. In *MobiSys*, 2004.
[28] A. Woo, S. Seth, T. Olson, J. Liu, and F. Zhao. A spreadsheet approach to programming and managing sensor networks. In *IPSN*, 2006.
[29] Y. Yao and J. Gehrke. The Cougar Approach to In-Network Query Processing in Sensor Networks. In *SIGMOD*, 2002.