A Spreadsheet Approach to Programming and Managing Sensor Networks

Alec Woo
Arched Rock Corp.
657 Mission St. Ste. 600
San Francisco, CA 94105
alecwoo@acm.org

Siddharth Seth
Dept. of EECE
Indian Institute of Technology
Kharagpur, India
siddharth.seth@iitkgp.ac.in

Tim Olson, Jie Liu, Feng Zhao
Microsoft Research
One Microsoft Way
Redmond, WA 98052
{timol,liuj,zhao}@microsoft.com

ABSTRACT

We present a spreadsheet approach to simplifying the process of managing, programming, and interacting with sensor networks and visualizing, archiving and retrieving sensor data. An Excel spreadsheet prototype has been built to demonstrate the idea. This environment provides Excel users, who are already familiar with spreadsheet applications, a convenient and powerful tool for programming and data analysis. We discuss the architecture of this prototype and our experience in implementing the tool. We show two different classes of sensor-net applications built using this platform. We also present performance data on the scalability of the tool with respect to data rate and number of data streams.

Categories and Subject Descriptors

H.4.1 [**Office Automation**]: Spreadsheets

General Terms

Experimentation

Keywords

Networked sensors, Excel, SQL server, data streams

1. INTRODUCTION

Today, one of the hurdles in deploying a sensor network is the difficulty of programming and running the entire system. It involves programming and managing the sensors, programming the gateways, interpreting and processing the data streams, and setting up servers for data archival and retrieval. The truth is that even experts in the field find this entire process difficult and troublesome.

The underlying challenges arise from the inherent distributed nature of sensor networks that cross multiple tiers of computing platforms: sensors, gateways, and servers. In addition, the unpredictability of the physical phenomenon being monitored often requires much iteration of data analysis and algorithmic or processing changes. This "in-situ" property is unique to sensor network since deployment-specific constraints can impact the entire programming design and process; the deterministic behavior generally assumed in traditional programming is rare here.

Recent research efforts in simplifying sensor programming have led to a few interesting macro-programming frameworks [8, 7, 9, 11, 5]. The general approach of these works focuses on designing a high-level language primitives that can abstract away programming individual sensor nodes and low-level system details. These abstractions build up the important foundation and representation for defining programming specifications that aim to span multiple nodes. The high-level description of logic is very useful for programming and managing sensor systems composed of many nodes. However, sensor-net programming involves hurdles beyond building richer abstractions. The "in-situ" nature of sensor networks requires a joint real-time data acquisition, processing and programming environment. System management support that allows users to (re-)configure and (re-)program part or entire sensor system is also important. Therefore, a sensor network programming environment requires three concurrent components: data acquisition and processing, in-situ programming, and run-time system management and reconfiguration support.

The contribution of this work is to allow users to achieve these three requirements using the familiar and widely used spreadsheet environment — Microsoft Excel^{TM}. Using spreadsheet to organize sensor data is not completely new. The work in [2] explores the use of a spreadsheet interface to help scientists to visualize data and perform some limited processing. The main idea is to provide a pivot-table interface so that users can visualize the data relationships from different perspectives. Instead of using Excel, they build a complete spreadsheet tool from scratch. The work also proposes to let users write simple expressions to define event triggers that will eventually be compiled down to the sensor-tier acting as low-level filters.

Our environment extends the popular office software package and builds upon the abstractions and run-time support developed by the macro-programming effort and the sensor-net tools like TinyOS[1]. The ultimate goal is to provide a simple user experience for interacting with sensor networks, in terms of both data management, and programming. Al-

[1]http://www.tinyos.net

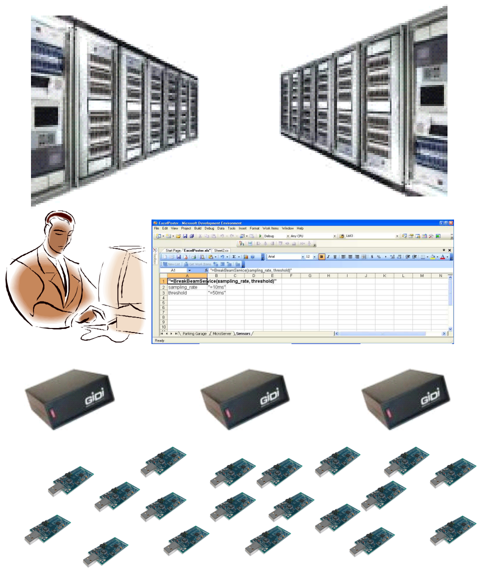


Figure 1: A multi-tier architecture for sensor network deployments.

though processing is currently performed centrally in the spreadsheets, it can be distributed in the future for scalability reasons with appropriate run-time support.

Our current implementation of the prototype with Excel is built on top of a service-oriented abstraction architecture provided by SONGS [6]. Although there are other ways to interface Excel with sensor networks, this service-oriented abstraction layer and the XML Web Service interface it provides are flexible and easy to work with. Its extensibility is also crucial when we distribute data processing in the future.

The paper is organized as following. Section 2 discusses how spreadsheet fits within the multi-tier architecture typically found in sensor network deployments. Section 3 walks through a high-level description of the implementation of our prototype and how it addresses our programming and management goals. Section 4 presents two application scenarios showing how our prototype helps building sensor-net applications. Section 5 evaluates the scalability of our spreadsheet prototype. We discuss our experiences in designing this toolkit in Section 6 and related work in Section 7, and we conclude and address future work in Section 8.

2. THE ROLE OF SPREADSHEET

Figure 1 shows the architectural overview of a sensor network deployment that is typically seen in scientific data collection, environmental monitoring, and event detection applications.

The lowest-tier of computing at the bottom of Figure 1 consists of sensors deployed over a field. The sensors self-organize into a single or multiple-hop network streaming data towards the gateways (also called *microservers* in some literatures), the next tier of computing. The gateway nodes may either be wireless or wired, but typically will have higher link bandwidth and better reliability. They can be programmed to further process the data streams or even operate as delegates for querying the sensor tier. The gateway, as its name suggests, also acts as an entry point to the Internet for the sensor nodes. The data streams from the gateways can be aggregated and archived for further processing by more powerful servers or databases at the next tier. This last tier of computing at the top of Figure 1 supports an Internet-scale sensor system. Managing and programming each of the tiers are difficult, not to mention integrating them together to form a complete system. Therefore, the design space crossing these different tiers is large and many of the issues are out of the scope of this paper. In this section, we focus on identifying the roles of spreadsheet relative to Figure 1.

Users typically interact with the sensor system off-line at the top tier or in real-time at the middle tier. Thus the boundary between the top and the middle tiers is particularly important. Spreadsheet can play the roles of simplifying data acquisition, data organization, programming, and run-time system management and reconfiguration interfaces for the users at the middle tier. Since it is already a familiar programming and data organization interface for scientific and business purpose, applying the same spreadsheet programming model for processing incoming data streams in real time would be very useful for everyday computer users who do not know or want to learn sensor network details. The same built-in statistical functions and data visualization tools in a spreadsheet can be reused for analyzing data streams in the sensor-net context.

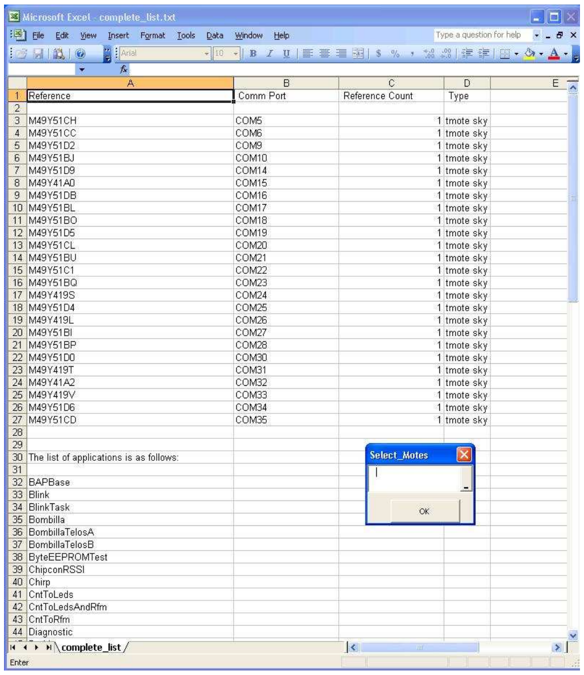
Another advantage of using the spreadsheet model is the easiness of combining streaming real-time data with static information such as node locations, configurations, and non-sensor related information. This is useful for system management and reconfiguration to have a customized view of various information sources. For example, one can overlay the deployment map and bind the spreadsheet cells to the sensor node positions on the map. This allows users to both manage the node and its data stream like manipulating individual cell on a spreadsheet. Changing the values in the cell of a node can automatically configure or even reprogram the node.

This integration of spatial deployment information with data analysis, data programming and system management directly addresses the “in-situ” nature of sensor networks. Spreadsheet naturally positions itself between the user and the gateway and sensor tier as shown in Figure 1. Furthermore, the ease of integration of spreadsheet with web services and databases at the highest-tier makes spreadsheet a very convenient integration tool. To preserve the distributed nature of the system, data processing within spreadsheets can fall back to other nodes, the gateways, or even the sensor-net tiers. However, this goes beyond the current scope of this paper.

3. SYSTEM IMPLEMENTATION

The implementation of the spreadsheet prototype has two parts. The first involves augmenting Excel for sensor network management. The second involves making Excel handle streaming data. In our prototype, we use Tmote sky from Moteiv[2] sensor nodes, PC gateways with a microserver runtime (called μSEE[3]), a user PC machine with Excel 2003 and Visual Studio 2005, and a Microsoft SQL Server.

[2]http://www.moteiv.com.
[3]Available at http://research.microsoft.com/nec/msrsense


Microsoft Excel - complete_list.txt
File Edit View Insert Format Tools Data Window Help
Type a question for help
Select_Motes
OK
complete_list
Enter

	A	B	C	D	E
1	Reference	Comm Port	Reference Count	Type	
2					
3	M49Y51CH	COM5	1	tmote sky	
4	M49Y51CC	COM6	1	tmote sky	
5	M49Y51D2	COM9	1	tmote sky	
6	M49Y51BJ	COM10	1	tmote sky	
7	M49Y51D9	COM14	1	tmote sky	
8	M49Y41A0	COM15	1	tmote sky	
9	M49Y51DB	COM16	1	tmote sky	
10	M49Y51BL	COM17	1	tmote sky	
11	M49Y51BO	COM18	1	tmote sky	
12	M49Y51D5	COM19	1	tmote sky	
13	M49Y51CL	COM20	1	tmote sky	
14	M49Y51BU	COM21	1	tmote sky	
15	M49Y51C1	COM22	1	tmote sky	
16	M49Y51BQ	COM23	1	tmote sky	
17	M49Y419S	COM24	1	tmote sky	
18	M49Y51D4	COM25	1	tmote sky	
19	M49Y419L	COM26	1	tmote sky	
20	M49Y51BI	COM27	1	tmote sky	
21	M49Y51BP	COM28	1	tmote sky	
22	M49Y51D0	COM30	1	tmote sky	
23	M49Y419T	COM31	1	tmote sky	
24	M49Y41A2	COM32	1	tmote sky	
25	M49Y419V	COM33	1	tmote sky	
26	M49Y51D6	COM34	1	tmote sky	
27	M49Y51CD	COM35	1	tmote sky	
28					
29					
30	The list of applications is as follows:				
31					
32	BAPBase				
33	Blink				
34	BlinkTask				
35	Bombilla				
36	BombillaTelosA				
37	BombillaTelosB				
38	ByteEEPROMTest				
39	ChipconRSSI				
40	Chirp				
41	CntToLeds				
42	CntToLedsAndRfm				
43	CntToRfm				
44	Diagnostic				

Figure 2: Downloading Tmote programs through Excel.

3.1 Network Management

We augment Excel with its built-in VBA scripting capability to provide a user interface that eases the process of programming (i.e. downloading code to) the sensor tier. The first step involves discovery of deployed sensor nodes. If nodes are connected directly to the USB port of the Excel PC, discovery simply means invoking corresponding TinyOS tools, such as "motelist", to query for the sensor nodes. If nodes are configured on an Ethernet network, through, for example, Tmote Connect, discovery often means going through a lookup table that contains information of all deployed sensor nodes. If nodes are deployed in a multi-hop mesh, we can rely on tools, such as Nucleus [10], for node discovery and display it in Excel.

Once node discovery is done, each sensor is represented as a cell, as shown in Figure 2. Users select the cells that represent motes for programming. All TinyOS applications in the `tinyos-1.x/apps` directory are shown as a list in Excel for users to select. Sending programs to the motes can be done through USB, Ethernet, or wireless (with e.g. Deluge [3]).

Once the motes and a program is selected, the VBA script will invoke external shell commands to recompile the program, to set the node IDs, and to download the program onto the corresponding motes.

3.2 Sensor Data Stream Processing

We use Visual Studio 2005 with .NET 2.0 to augment Excel to act as an agent to query the gateway tier for incoming sensor data streams. This version of Visual Studio integrates well with Excel development using `Visual Studio Tools for Office System`. This Visual Studio tool allow us to implement an extension of Excel in C# to handle streaming data.

Figure 3 shows the key components in implementing sen-

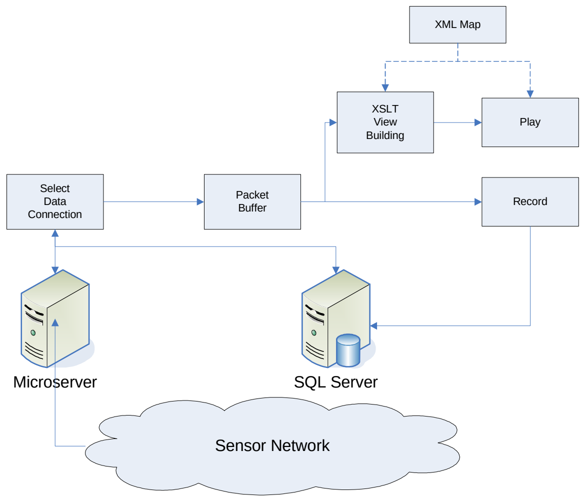


Figure 3: The flow chart of a typical data processing application.

sor data stream processing applications. Sensor nodes communicate with microserver gateways, which feed data into Excel. Using an action panel, users can select data sources — either real-time data from the sensor network, or archived data from a SQL server. Streams of data are first queued by a packet controller in a buffer. A separate display thread periodically takes data out from the buffer and concatenates multiple packets into a data view using XSL transformations, which assigns each data stream a unique namespace. The view is then displayed using the built-in XML mapping feature.

The gateway tier runs a service-oriented microserver, called μSEE. The microserver provides a service abstraction over the gateway and sensor tiers. A XML script called MSTML is used by μSEE for defining service composition over the gateway and the sensor tiers. Our current implementation uses a pre-defined MSTML file to task the two tiers. In the future, the construction of the service composition can also be done within Excel.

The microserver uses XML as the basic gluing logic and structure definition for service composition and data stream objects. That is, the gateways have services to automatically convert data streams from the lower-tier, platform-dependent sensor data format into platform-independent XML streams. Here is an example of the XML data for `Oscilloscope` messages produced by the gateways:

```
<?xml version="1.0" encoding="utf-8"?>
<ArrayOscopeMsg
xmlns:xsd= "http://www.w3.org/2001/XMLSchema"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
   <sourceMoteID>236</sourceMoteID>
   <lastSampleNumber>13782</lastSampleNumber>
   <channel>2</channel>
   <data>
      <unsignedShort>474</unsignedShort>
      <unsignedShort>475</unsignedShort>
      ...
      <unsignedShort>472</unsignedShort>
   </data>
</ArrayOscopeMsg>
```

The schema for this XML stream is then mapped to Excel cells using the built-in XML Map feature. XML mapping is a process of dragging and dropping a particular element from XML schema on to a cell, which informs Excel to bind data associated with that element to the cell upon loading future data. A limitation of this approach is that each element mapping must be unique in the XML Map definition. For this reason, we extended Excel by representing the data for each data stream as a unique namespace.

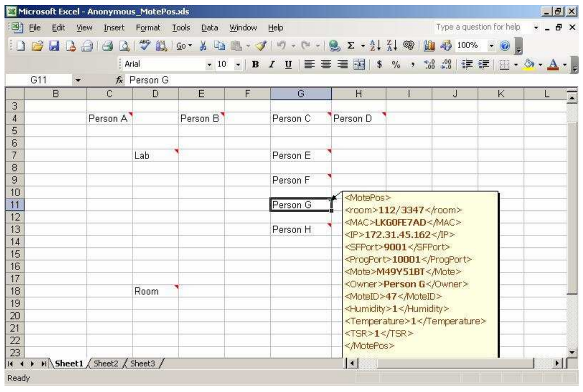
The usage of this tool requires no special knowledge of the packet stream or of XML maps as the user only needs to load the Excel worksheet, define the XML maps to use by loading a network configuration file, which automatically creates the necessary XSL transformations for the data streams. Then, the user would utilize the built-in XML Maps feature of Excel to drag elements from the list to the appropriate cell where they would like to bind the data. For any elements that have an unbounded size attribute, our extension to Excel will automatically grow lists when new data is added to the document. Once the maps are defined, a data connection is created by either specifying a real-time connection to the gateway or to SQL Server for previously recorded data. Once data streams are mapped onto the Excel lists, processing on the data is the same as usual spreadsheet programming. Notice that Excel has an event-driven model of computation built in. Whenever the value of a cell changes, all dependent formulas and plots are reevaluated and refreshed automatically. This allow uses to always have the most recent view of the processing results without managing data propagation manually.

3.3 Database Server

While spreadsheet provides a data programming and visualization environment to the users, the archival of the data requires a database, especially when data streams are continuous for a long period of time.

Our prototype can store each data stream and its corresponding processed values into the database. To ease the integration effort between Excel and the database, we use SQL Server 2005 because it natively supports XML data types. With this XML capability, XML streams can be stored directly in the database without knowing the packet structure a priori. XML query processing is thus possible over both the raw data streams and the processed data streams from the spreadsheet. We use XML as the key of simplifying the integration process for the different tiers as we believe that future databases and data streams will widely support and exploit XML.

With a database, the spreadsheet becomes a cache for the most recent data streams. This cache size is especially limited if processing is done centrally. In our current implementation, Excel lists of the data and processing streams have a limited size defined by the user. That is, each list signifies a fixed time window of the most recent data, with the current sample being at the bottom of the list. The stale samples, already stored in the database, will be automatically purged by Excel as the lists are bounded by their maximum length. The choice of this size is a balance on the amount of history that it needs to maintain for processing versus scalable performance as the amount of streaming increases. This size and the refresh rate are two key performance metrics to control the scalability of our tool.



```
<MotePos>
<room>112/3347</room>
<MAC>LKG0FE7AD</MAC>
<IP>172.31.45.162</IP>
<SFPort>9001</SFPort>
<ProgPort>10001</ProgPort>
<Mote>M49Y51BT</Mote>
<Owner>Person G</Owner>
<MoteID>47</MoteID>
<Humidity>1</Humidity>
<Temperature>1</Temperature>
<TSR>1</TSR>
</MotePos>
```

Figure 4: A spreadsheet illustration of sensor locations.

4. APPLICATION SCENARIOS

We apply our prototype to two different classes of sensor-net applications. One involves event detection for the presence of a vehicle in a parking lot and the other is a typical environmental data collection application within an office building.

4.1 Data Collection

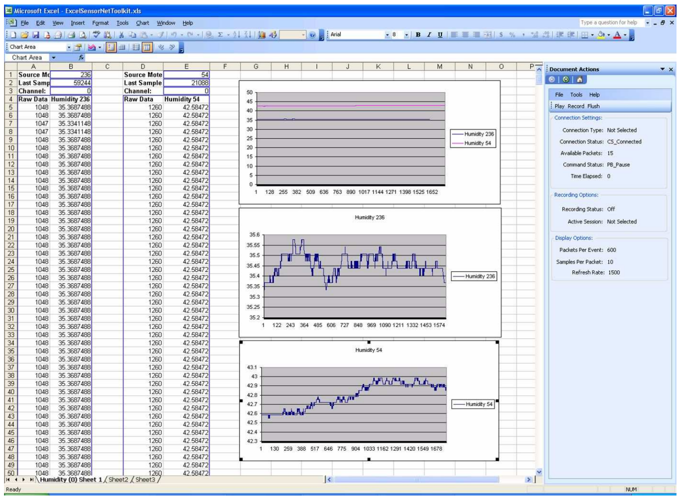
Our first application involves collecting environmental data within a typical office building. The deployment consists of 9 sensor nodes that are capable of sensing ambient light, photo-synthetic light, temperature, and humidity. These sensor nodes are deployed across seven offices and a hardware laboratory. We use Excel to maintain the spatial information of the deployment as shown in Figure 4.

Deployment specific information of each node can be described using XML and stored as cell values in the spreadsheet. The information may include node ID, types of sensors on the node, and location of the node. Depending on how the sensor program is created, this information can directly be mapped to Nucleus in TinyOS or abstracted by a SONGS service for management and reconfiguration purposes. The tool-tip box in Figure 4 shows the XML descriptions for one of our deployed node.

The XML elements, including HOST, IP, SFPort, ProgPort, and MoteSerial, describe the test-bed information about the mote. The rest of the elements specifically describe the deployment information and the services provided by the sensors. The deployment information is expected to be defined by the users while the node services can also be dynamically discovered.

To visualize and process the sensor data streams with Excel, users can select a subset or all of the sensors discovered and shown on the spreadsheet. The users can simply select the cells corresponding to the nodes to subscribe for their data streams. The selected data streams will be shown automatically on a different worksheet.

Excel can automatically infer the schema of incoming XML objects and display them on the spreadsheet. However, this is often not desired because users may want to have the control to layout and select important fields from the XML data streams. Our current prototype manually filters for the interesting part of the data streams and lays them out on the spreadsheet from left to right, with user-defined processing



	A	B	C	D	E
1	Source Mc	236		Source Mote	54
2	Last Samp	59244		Last Sample	21088
3	Channel:	0		Channel:	0
4	Raw Data	Humidity 236		Raw Data	Humidity 54
5	1048	35.3687488		1260	42.58472
6	1048	35.3687488		1260	42.58472
7	1047	35.3341148		1260	42.58472
8	1047	35.3341148		1260	42.58472
9	1048	35.3687488		1260	42.58472
10	1048	35.3687488		1260	42.58472
11	1048	35.3687488		1260	42.58472
12	1048	35.3687488		1260	42.58472
13	1048	35.3687488		1260	42.58472
14	1048	35.3687488		1260	42.58472
15	1048	35.3687488		1260	42.58472
16	1048	35.3687488		1260	42.58472
17	1048	35.3687488		1260	42.58472
18	1048	35.3687488		1260	42.58472
19	1048	35.3687488		1260	42.58472
20	1048	35.3687488		1260	42.58472
21	1048	35.3687488		1260	42.58472
22	1048	35.3687488		1260	42.58472
23	1048	35.3687488		1260	42.58472
24	1048	35.3687488		1260	42.58472
25	1048	35.3687488		1260	42.58472
26	1048	35.3687488		1260	42.58472
27	1048	35.3687488		1260	42.58472
28	1048	35.3687488		1260	42.58472
29	1048	35.3687488		1260	42.58472
30	1048	35.3687488		1260	42.58472
31	1048	35.3687488		1260	42.58472
32	1048	35.3687488		1260	42.58472
33	1048	35.3687488		1260	42.58472
34	1048	35.3687488		1260	42.58472
35	1048	35.3687488		1260	42.58472
36	1048	35.3687488		1260	42.58472
37	1048	35.3687488		1260	42.58472
38	1048	35.3687488		1260	42.58472
39	1048	35.3687488		1260	42.58472
40	1048	35.3687488		1260	42.58472
41	1048	35.3687488		1260	42.58472
42	1048	35.3687488		1260	42.58472
43	1048	35.3687488		1260	42.58472
44	1048	35.3687488		1260	42.58472
45	1048	35.3687488		1260	42.58472
46	1048	35.3687488		1260	42.58472
47	1048	35.3687488		1260	42.58472
48	1048	35.3687488		1260	42.58472
49	1048	35.3687488		1260	42.58472
50	1048	35.3687488		1260	42.58472

Figure 5: A spreadsheet data collection scenario.

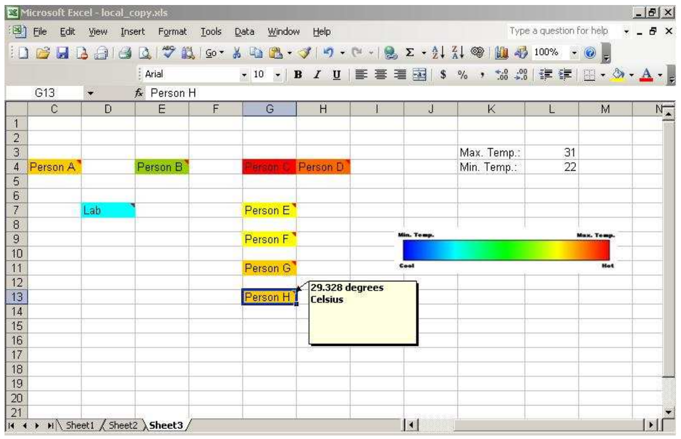


Figure 6: Real-time sensor data overlayed on sensor locations.

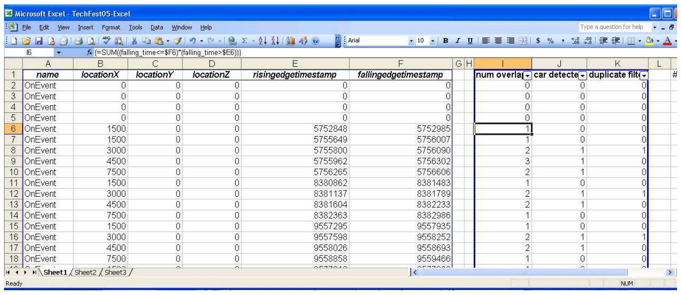
lists interleaving between them. In the future, we plan to use a wizard to guide users to control such a process.

Figure 5 shows an example of the filtered real-time data streams from two sensor nodes in Figure 4, with the most recent samples located at the bottom of each list. In this case, each processing list simply converts the unit of raw humidity data from the list on its left into Celsius.

Users can utilize Excel's built-in plotting ability to graph real-time statistics of the data streams similar to the Oscilloscope tool from the TinyOS distribution. In addition, users can reuse the deployment specific information layout in Figure 4 to visualize the spatial distribution of the data. For example, Figure 6 shows a very simple spatial contour graph of the temperature variations among the different locations. The color spectrum range is adjusted dynamically, depending on real-time data as shown in cells `L3` and `L4`. Clicking on the node will display a callout box showing the average temperature of that location.

4.2 Vehicle Event Detection

The second application involves detecting vehicles and counting them within a parking lot infrastructure. We use



`I6` `{=SUM((falling_time<=$F6)*(falling_time>$E6))}`

	A	B	C	D	E	F	G	H	I	J	K
1	name	locationX	locationY	locationZ	risingedgetimestamp	fallingedgetimestamp			num overla	car detecte	duplicate filt
2	OnEvent	0	0	0	0	0			0	0	0
3	OnEvent	0	0	0	0	0			0	0	0
4	OnEvent	0	0	0	0	0			0	0	0
5	OnEvent	0	0	0	0	0			0	0	0
6	OnEvent	1500	0	0	5752848	5752985			1	0	0
7	OnEvent	1500	0	0	5755649	5756007			1	0	0
8	OnEvent	3000	0	0	5755800	5756090			2	1	1
9	OnEvent	4500	0	0	5755962	5756302			3	1	0
10	OnEvent	7500	0	0	5756265	5756606			2	1	0
11	OnEvent	1500	0	0	8380862	8381483			1	0	0
12	OnEvent	3000	0	0	8381137	8381789			2	1	1
13	OnEvent	4500	0	0	8381604	8382233			2	1	0
14	OnEvent	7500	0	0	8382363	8382986			1	0	0
15	OnEvent	1500	0	0	9557295	9557935			1	0	0
16	OnEvent	3000	0	0	9557598	9558252			2	1	1
17	OnEvent	4500	0	0	9558026	9558693			2	1	0
18	OnEvent	7500	0	0	9558858	9559466			1	0	0

Figure 7: A spreadsheet program detection vehicle events.

the same deployment setup as discussed in [6]. Infrared break-beam sensors and their reflectors are deployed in the parking lot. Whenever the line of sight between a sensor and its reflector is physically blocked by an obstacle, such as a vehicle, the sensor will report a low-level binary detection event to the gateway. A vehicle passing over these sensors will generate streams of events, which are time-stamped by the gateway and sent to Excel. Figure 7 shows a screenshot of the data streams in Excel.

In this case, an "OnEvent" means that the break-beam sensor has an event detected. The XML representation for OnEvent contains the location of the break-beam sensor and the time stamps for the rising and falling edges. In our deployment, sensors are strategically placed along the roadside such that a typical vehicle would block at least two of the sensors during crossing. One simple mechanism in detecting a vehicle is to count if two or more sensor events within a given physical location overlap in time. The list on the right with a thick border in Figure 7 shows how users can specify such processing using conventional spreadsheet array formula.

For each raw data detection event, we calculate the number of events that overlap with it in time using the following Excel array formula: {`=SUM((falling_time<=$F6)*(falling_time>$E6))`}. In this case, the current row is 6. Therefore, `F6` represents the falling-edge timestamp and `E6` represents the rising-edge timestamp of the current OnEvent. Variable `falling_time` is the name of column F defined by the user a priori. It signifies the falling-edge timestamp of all the events in the worksheet. The action of the formula is to sum up the number of events in the list that has the falling-edge timestamps within the rising-edge and falling-edge of the current event at row 6. This is an array formula, as denoted by {}, because it has to process the entire `falling_time` (F^{th}) column. Expanding the formula to include location information in Figure 7 can filter out events that are not spatially correlated. As the raw data list grows, the processing list grows as well with the relative cell addressing, such as F6 and E6, in the formula adapting automatically.

The next column defines a threshold for vehicle detection; if the number of overlapping detections equals or exceeds a threshold of 2, a vehicle is detected. This logic is translated to the following Excel expression in cell J6 in Figure 7: `=IF(num_overlaps>=2, 1,0)`. This is not an array formula since we are not processing the entire `num_overlaps` column. Variable `num_overlaps` simply refers to the cell in column (I) on the same row numbered 6.

This simple example demonstrates how users can use the built-in Excel functions and expressions to derive more high-level statistics, such as vehicle speed and traffic flow information within the parking lot. The naming convention in Excel can be confusing when writing formulas since it depends on the context of the formula, which is one limitation of the Excel programming model.

Notice that incoming raw data streams do not follow a temporal order in Figure 7. However, the amount of buffer (list size), which determines how far back the spreadsheet can go in time, is adequate to absorb these issues for our simple application. However, users need to explicitly process the unordered streams if tight temporal processing is required.

5. PERFORMANCE EVALUATION

Key questions to ask about the scalability of this Excel toolkit are how many concurrent input streams and at what rates it can process data received from a sensor network. To evaluate the performance, we use a data collection example as described in Section 4.1. by running the application under a various number of data streams.

We programmed a set of motes with the `Oscilloscope` application that comes with TinyOS-1.1.13 distribution. Data are sent at roughly one packet per second per channel. Each packet contains 10 samples of unsigned 16-bit integers. A microserver gateway converts received TinyOS packets to XML packets as shown in section 3.2. On a desktop PC with Intel P4 2.80 GHz CPU and 2GB of RAM, we run our Excel spreadsheet that displays all incoming data, one sample per cell. Each sensor channel is a separate column in the spreadsheet.

The execution time is measured using the `StopWatch` class in the .NET 2.0 Framework. This class allows us to capture elapsed time for a code section. The primary code analyzed is the display thread for each stream. The activation of the thread is controlled by a refresh rate parameter as described below. Upon activation, it takes a fixed number of packets from the packet buffer, applies the XSL transformation to merge them into a single XML document with a unique namespace, and display the data in Excel GUI through XML binding. There is no further processing or plotting of the data in Excel. So, essentially, we only collect the overhead of consuming raw data into Excel from streams. We call this the *processing time* in the following discussions.

There are several parameters that users can control how data are consumed by Excel. They are keys to understand the performance metrics:

- Timeline (T) – The timeline, or window size, is the maximum number of packets displayed in Excel for a particular stream. For example, if the timeline is 300 packets and the data are coming at a rate of one second per packet with 10 samples per packet, then Excel roughly caches 5 minutes of data that occupies 3000 cells. Once the timeline is reached, the packet controller removes old packets from the beginning of the buffer and adds new packets to the end, essentially sliding the time window in the direction of moving time.

- Refresh rate (RR) – The refresh rate, measured in number of packets per refresh, is used by the packet controller to notify the display thread to consume latest data and to update the display. For example, $RR = 1$ is the fastest refresh rate, such that the GUI responds to every incoming packet. If $RR = 10$, then the packet controller will wait until receiving 10 new packets before notifying the display thread. Since incoming packets are queued in the packet buffer, batch-processing them with a low refresh rate (or a high packet number) can give Excel more time to transform and display packets, thus improve scalability.

- Sub-packet – Sometimes, a user may only want to display a subset of data contained in the receiving packets. For example, Oscilloscope packets in our example has 10 samples in each input packet, this parameter allows users to downsample them in the user interface, even though all samples are archived in the database. We use the parameter samples-per-packet-to-display (SPP) to control how much data to display in Excel. SPP=10 means all samples are displayed, while SPP=1 means only one sample per packet is displayed.

Figure 8 shows the processing time of various numbers of packets. The refresh rate is 1 and SPP is 10. The curve is the same under different system load, varying from 1 to 9 data streams. The line in the plot is $y = x$, in order to illustrate that the processing time is nearly a slow exponential with respect to the length to the timeline.

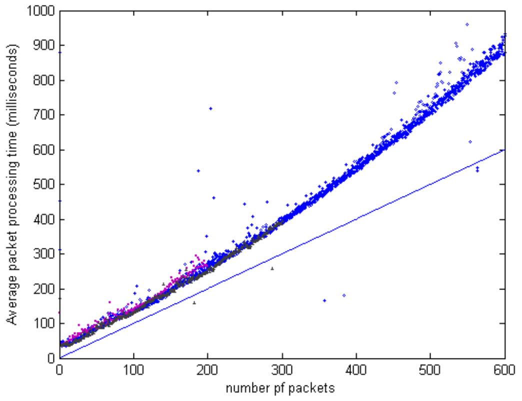


Figure 8: Average processing time over the number of XML packets to be binded, $RR = 1$, $SPP = 10$.

Figure 9 shows a histogram of the processing time for 600 packets in one stream with $RR = 1$ and $SPP = 10$, which corresponds to 6000 rows in the spreadsheet. The histogram plots 2000 runs. We see the same statistics in 8 hours of stress test. So, it takes Excel roughly a second to process 600 packets, which is about 10 minutes of raw oscilloscope samples with 10Hz sampling rate.

The data also indicates that when Excel is processing more than one data streams, the total timeline has to be shorter, otherwise the receiving packets will accumulate in the packet buffer and Excel loses real time property. We have observed that when the packets in the buffer accumulate too much, the Excel UI becomes non-responsive. To avoid this problem, one can reduce the refresh rate or reduce the samples per packet to be displayed.

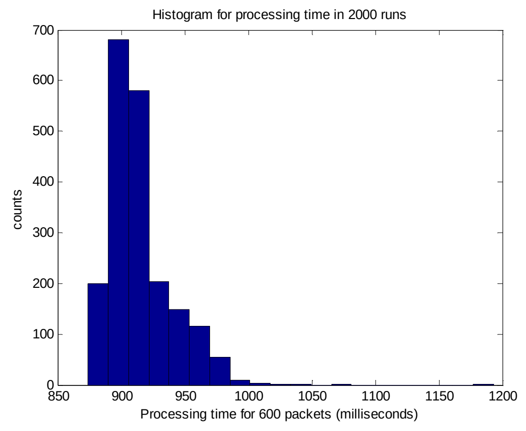


Figure 9: A histogram of the processing time for 600 packets, $RR = 1$, $SPP = 10$.

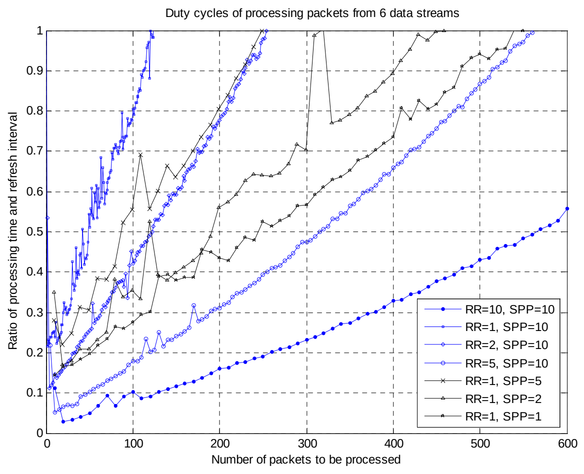


Figure 10: The effectiveness of adjusting refresh rate and samples to display parameters.

Figure 10 shows the effectiveness of adjusting RR and SPP. We define

$$\text{DutyCycle} = \frac{\text{processing time per refresh}}{\text{refresh interval}}.$$

The plot shows the duty cycles for processing 6 streams of input under either slowing down the refresh rate (to 2, 5, or 10 packets/refresh) or down sampling the packets (to 5, 2, and 1 samples per packet to display). It is clear that adjusting refresh rate is more effective than down-sampling the packets. In fact, the effect of changing refresh rate is almost linear. This is due to the savings on redrawing all the cells in Excel.

We have also stress tested the robustness of the tool by running it for days and over various mote connection mechanisms: through local USB ports, through local area network, and through corporate intranet. We have not found any memory management problems in handling long streams.

Consider the rate that human consumes information in data streams, our Excel prototype is scalable to meet most users' requirements. At a once per 10 seconds refresh rate, it can easily display 6000 rows from a single data stream. There is enough time left to perform many further processing and plotting.

6. DISCUSSIONS

Our prototype allows us to gain a lot of first-hand experience in evaluating this spreadsheet approach to sensor networks. We discuss our programming and implementation experiences in this section.

6.1 Programming Experiences

For developers who are used to imperative languages for programming, spreadsheet programming may feel restrictive in the beginning. It is easy to arrive with an early judgment that “this can't be done in a spreadsheet!” This is because spreadsheet may provide less flexibility with its simple expression language and its requirement of the data to be in a tabular format. For sophisticated data processing, which we haven't evaluated with our two application scenarios, spreadsheet programming may be cumbersome. For example, list-to-list processing primitives, which are very common for data streams, are not supported by Excel. Data naming is also weak as the reference of the list's name depends on the context of the formula. While VBA can be used to provide a rich functional language programming platform, it breaks the spreadsheet programming model. Non-VBA functional programming on Excel is possible as demonstrated by the work in [4], but it is still an early research prototype. At the current stage, we have not explored using spreadsheet for programming complex processes.

However, the very point of a spreadsheet is to provide a simple programming platform for non-computer scientists, who will be the main users of sensor networks. Although our application scenarios are simple, the demonstrated processing is fundamental and common to many potential sensor-net applications. Furthermore, the formula expression language in Excel is fairly general and supports a rich set of statistical functions for scientific studies. Finally, the programming logic and the associated buffer usage are quite explicit on a spreadsheet. This can be advantageous since such information is useful in delegating the processing to the same or lower tiers.

6.2 Implementation Experiences

We found that the built-in VBA capability in Excel provides a simple way for users to customize user interfaces and automate data processing for each application, which is important since sensor network is often application-specific.

Our design choice of relying on XML for integrating the different pieces of the system together is crucial as it allows us to decouple the implementation platform details among the individual pieces and thus, simplify the process of integration. The trend that business logic integrations and Internet applications are converging to web services is clear. We believe that the same trend will penetrate the sensor network deployments so that static and dynamic data can be easily combined.

The Visual Studio Tools for Office Systems is very useful to our implementation. There are numerous built-in optimizations in Excel and .NET2.0 for processing XML documents. It allows us to easily build a prototype without getting Excel source code. We can treat Excel components as an object library and implement our extension in C#. Our extension mainly consists of socket interfaces for communication with the gateway and with the SQL server.

7. RELATED WORK

The idea of using the tabular or spreadsheet interface for managing and processing data is not new; it is used in everyday business, scientific, and industrial settings. It is this reason why we advocate the spreadsheet approach to managing and programming sensor networks. Our contribution is in developing a prototype that demonstrates the potential of this approach for "in-situ" sensor data management. Several other research works have taken similar paths to simplify sensor data processing and visualization.

TinyDB [7] provides the tabular interface for users to visualize collected sensor data. It presents a SQL query interface for data processing and provides a primitive tool for data visualization. Its main goal is to perform in-network processing in the sensor tier in response to a given SQL query. Tinker [1] is a methodology and framework for data centric sensor network applications. The Excel data processing interface can be a front end for Tinker.

Moteview from Crossbow[4] is a sensor network management and data collection tool. It supports visualization of the physical deployment and maps it with the corresponding sensory data. The tool is oriented towards visualization of sensor data and network statistics such as network topology. It does not present a programming interface for users to process the incoming data streams.

The Nucleus management console project at Berkeley[5] provides a spreadsheet web interface for showing system information of each sensor node provided it has the TinyOS Nucleus run-time installed. The system information can represent networking statistics or even the internal states of each node. The console provides a tabular interface for users to view and alter such information, thus, performing system-wide management and reconfiguration support.

8. FUTURE WORK AND CONCLUSION

We have successfully built a spreadsheet prototype in Excel to address three important issues of sensor-net "in-situ" programming, which include real-time data programming, data analysis, and system management and reconfiguration. We built two different classes of applications to demonstrate the feasibility of the spreadsheet approach and presented our experience.

Systematic performance and robustness studies on the database connections and full system integration are necessary. They allow us to explore the crossover point in offered load between centralized and distributed stream processing with Excel. We plan to do such evaluation using our testbed and grow the scale incrementally.

Beside performance and stability, we should conduct usability study of our prototype to guide us on the interface design. We also plan to build other applications to drive the further development of the tool.

We only explored some of the data analysis, programming and system management issues with our current prototype. A few interesting and important directions remain to be explored. They include user-interface design on manipulating many data streams, new ways of data and network visualization, identification of useful statistical functions, and process delegation and service composition for the gateways and sensor-tier.

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

[5]http://www.tinyos.net/ttx-02-2005/developments/Nucleus%20NMS.ppt

9. REFERENCES

[1] Elson, J., and Parker, A. Tinker: A tool for designing data-centric sensor networks. In *Fifth Intl. Conf. on Information Processing in Sensor Networks (IPSN 2006), SPOTS track. Nashville, TN.* (April 2006).

[2] Horey, J., Bridges, P., Maccabe, A., and Mielke, A. Work in Progress: The design of a spreadsheet interface for sensor networks. In *Proc. of Information Processing in Sensor Networks (IPSN'05), Los Angeles, CA* (April 2005).

[3] Hui, J. W., and Culler, D. The dynamic behavior of a data dissemination protocol for network programming at scale. In *Proc. 2nd ACM Conference on Embedded Networked Sensor Systems (SenSys'04), Baltimore, MA* (November 2004).

[4] Jones, S. P., Burnett, M., and Blackwell, A. A user-centred approach to functions in excel. In *Proc. of Intl. Conf. on Functional Programming (ICFP'03), Uppsala* (Sept 2003), pp. 165–176.

[5] Liu, J., Chu, M., Liu, J., Reich, J., and Zhao, F. State-centric programming for sensor-actuator network systems. *IEEE Pervasive Computing 2*, 4 (Oct–Dec 2003), 50–62.

[6] Liu, J., and Zhao, F. Towards semantic services for sensor-rich information systems. In *Proc. of the 2nd Intl. Workshop on Broadband Advanced Sensor Networks (Basenets'05), Boston, MA* (October 2005).

[7] Madden, S. R., Franklin, M. J., Hellerstein, J. M., and Hong, W. TAG: a Tiny AGgregation Service for Ad-Hoc Sensor Networks. In *OSDI* (December 2002).

[8] Newton, R., and Welsh, M. Region streams: Functional macroprogramming for sensor networks. In *Proceedings of the First International Workshop on Data Management for Sensor Networks (DMSN), Toronto, Canada* (Augest 2004).

[9] Ramakrishna Gummadi, Omprakash Gnawali, R. G. Macro-programming wireless sensor networks using Kairos. In *Proc. of the Intl. Conference on Distributed Computing in Sensor Systems (DCOSS), Marina del Rey, CA* (June 2005).

[10] Tolle, G., and Culler, D. Design of an application-cooperative management system for wireless sensor networks. In *Proc. 2nd European Workshop on Wireless Sensor Networks (EWSN), Istanbul, Turkey* (January 2005).

[11] Whitehouse, K., Zhao, F., and Liu, J. Semantic streams: a framework for the composable semantic interpretation of sensor data. In *Proc. European Workshop on Wireless Sensor Networks (EWSN'06), Zurich, Switzerland* (Feb. 2006).