SAMPL: A Simple Aggregation and Message Passing Layer for Sensor Networks

(Invited Paper)

Anthony Rowe Karthik Lakshmanan Ragunathan (Raj) Rajkumar
Electrical and Computer Engineering Department
Carnegie Mellon University
Pittsburgh, PA. USA.
{agr,klakshma,raj}@ece.cmu.edu

ABSTRACT

In recent years, wireless sensor networking has shown great promise in applications ranging from industrial control, environmental monitoring and inventory tracking. Given the resource-constrained nature of sensor devices and the dynamic wireless channel used for communication, a sensor networking protocol needs to be compact, energy efficient and highly adaptable. In this paper we present SAMPL, a simple aggregation and message passing layer, aimed at flexible aggregation of sensor information over a long period of time, and supporting sporadic messages from mobile devices. SAMPL is a compact network layer that operates on top of a low-power CSMA/CA based MAC protocol. The protocol has been designed with extensibility in mind to support new transducer devices and unforeseen applications without requiring reprogramming of the entire network. SAMPL uses a highly adaptive tree-based routing scheme to achieve highly robust operation in a time-varying environment. The protocol supports peer-to-peer data transactions, local storage of data similar to what many RFID systems provide as well as secure gateway to infrastructure communication. SAMPL is built on top of the Nano-RK[1] operating system that runs on the FireFly sensor networking platform. Nano-RK's resource management primitives are used to create virtual energy budgets within SAMPL that enforce application lifetimes. As of October 2008, SAMPL has been operating as part of the Sensor Andrew project at Carnegie Mellon University with battery powered sensor nodes for over seven months and continues to be actively used as a research testbed. We describe our deployment tools and network health monitoring strategies necessary for configuring and maintaining long-term operation of a sensor network. Our approach has led to sustainable average packet success rate of 94% across the entire network.

WICON 2008 November 17-19, 2008, Maui, Hawaii, USA.

Categories and Subject Descriptors

C.2.0 [**Computer-Communication Networks**]: Sensor Networks

General Terms

Wireless Sensor Networks

Keywords

Sensor Networks, Tree Routing, Deployment, Network Management

1. INTRODUCTION

Wireless sensor networks have generated significant research interest in recent years, due to their potential to seamlessly study the physical world. Despite the wealth of existing literature in the domain, very few real-world deployments have lasted more than a few months. Limited flexibility, energy constraints, lack of visibility, interaction complexity and maintenance overheads, still pose significant problems towards developing practical wireless sensor network deployments. In this paper, we address these issues and describe our simple aggregation and message passing layer (SAMPL) for wireless sensor networks, which has been operational for well over seven months, and still continues to successfully serve as a large living testbed for sensor networks research at Carnegie Mellon University.

Energy-efficient MAC protocols and routing algorithms have received considerable attention from the sensor networking research community. The underlying problem of time-varying channels and unreliable nature of wireless communication, continue to plague advances towards a practical solution. One of our key goals is to overcome this issue, and hence our design choices are towards developing a highly dynamic, flexible, yet energy-efficient solution to collect sensor data from the environment. We are able to achieve these properties by focusing our efforts on data aggregation, a first class primitive for most long-term sensing deployments.

Reconfigurability is key to developing a successful sensor networking deployment. Most wired sensor networking deployments like video surveillance systems, fire monitoring systems, building temperature control systems etc., have operator consoles from which administrators can configure the entire network operation. Similar requirements also exist for continuous sensing systems in the wireless domain. The network update rates should be increased/decreased on

Digital Object Identifier: 10.4108/ICST.WICON2008.4929
http://dx.doi.org/10.4108/ICST.WICON2008.4929

demand, and the system should re-configure on-the-fly to adapt to such changing requirements. For instance during a fire emergency, the operator may be interested in sensing the increasing CO_2 levels and temperature conditions at a much higher rate, compared to normal operating conditions. SAMPL is fully integrated with a complete configuration and control layer, which enables on-demand updates to all protocol parameters and participating sensor nodes.

Enabling out-of-the-box solutions to sensor networks involves developing practical solutions to deploying large-scale sensor networks. Although similar problems are regularly faced in the 802.11 domain, the multi-hop nature of wireless sensor networks, and their limited diagnostic capabilities, pose significant challenges for novice users. We have therefore developed a portable deployment device with a LCD display and easy-to-use user interface, to assist in deploying wireless sensor nodes. Our deployment strategy enables the users to ensure sufficient coverage, and significant redundancy, without getting involved in system details. Network extensibility is a key feature of our solution. Since the network was initially deployed, we have continued to add new sensor nodes, support newer types of sensor nodes and develop new applications, without requiring to modify any of the underlying system-wide protocols.

Debugging large scale sensor network deployments is a huge maintenance nightmare. This is one of the key reasons for failure behind most long-term deployments. We have therefore developed a continuous passive network monitoring framework, to study the behavior of our network over extended periods of time. We also instrumented some of our nodes with additional higher bandwidth network interfaces like 802.11 and powered them, to collect significant amounts of debugging information over our long period of deployment. This information is vital to our understanding of the protocol behavior, and helps guide future development endeavors for newer versions of SAMPL.

1.1 Protocol Goals

The SAMPL protocol is a compact network layer that operates on top of a low-power MAC layer to achieve robust and economic aggregation of network-wide sensor information. Although our current implementation uses a low-power listen protocol similar to BMAC[2], the same approach is also applicable to other energy-efficient MAC protocols. The protocol has been designed with extensibility in mind to support new transducer devices and unforeseen applications without requiring reprogramming of the entire network. The goals of the SAMPL protocol are as follows:

1. Efficiently collect sensor data using packet aggregation
2. Provide a control channel from the gateway to all nodes
3. Infrastructure support for mobile devices
4. Remote storage on each node similar to active RFID
5. Secure Communication
6. Built-in debugging for high visibility

1.2 Related Work

One of the early long-term wireless sensor network deployments was the Great Duck Island project [3]. This deployment of 43 motes lasted for about four months, and collected more than a million data points. The network was deployed in a remote island off the coast of Maine, to perform habitat monitoring for biological research. Although the deployment is comparable in duration to ours, we are concerned with an indoor office environment, where there is significantly more interference and time-variance in the environment. Our design choices, experiences and inferences are closely related to the future challenges for deploying sensor networks in buildings and work environments. Extensibility is also another key distinction of our work. We have continued to add sensor nodes, support newer types of sensors, and develop applications, since our initial deployment, without having to modify any part of our protocol infrastructure.

Zebranet [4] is another sensing project undertaken to study wildlife behavior in Kenya. The underlying challenges and solutions are very distinct from our scenario. Our nodes are much smaller in size, and need to integrate seamlessly with the environment. In Zebranet, the nodes are much larger in size, however they have to deal with additional challenges of mobility and wear-and-tear. Our deployment goals are also much more longer term in nature, and hence add additional constraints on our sensing infrastructure.

Significant efforts have been devoted to developing energy-efficient MAC protocols [5, 6, 7, 8, 9, 10, 11]. We use a variant of the low-power listen MAC protocol, which resembles the BMAC [2] protocol. Researchers have also attempted to narrow the waist-line of sensor networks, and develop a standardized IP-style network protocol [12, 13]. A marked deviation from such generalized approaches is to consider data aggregation as a fundamental service of sensor networks.

The TAG [14] service represents one of the early endeavors to restrict attention to sensor data aggregation. Although our approach shares the same ideals, we are forced to provide more flexibility and extensibility in our network. We support a varied set of services such as peer-to-peer messaging, asynchronous upstream Internet traffic, secure communication support, in-network re-configuration, and continuous diagnostics. It would be arbitrarily complex, if not impossible, to support such a diverse set of services using the SQL-like interface provided by TAG.

Researchers have also exclusively looked at developing mobile sensor nodes [15, 16, 17, 18]. Our mobile node is built specifically for aiding in network deployment, and enabling an out-of-the-box solution. In the future, we would like to extend the capabilities of this device, however maintaining its primary purpose of acting as a deployment device. Previous research has also looked at understanding and characterizing packet loss statistics [19, 20]. Our results are based on long-term measurements from an actual deployment, and are hence expected to be a much more faithful representation of the real-world conditions.

1.3 Paper Organization

The rest of the paper is organized as follows. Section 2 describes the SAMPL protocol. Section 3 outlines a deployment strategy used to optimize the initial placement of nodes. Section 4 shows a simple but effective statistical monitoring technique that can identify problem areas in the network. Finally, Section 5 provides concluding remarks and future work.

2. SAMPL PROTOCOL DESIGN

SAMPL uses a tree-based routing scheme in which con-

Digital Object Identifier: 10.4108/ICST.WICON2008.4929
http://dx.doi.org/10.4108/ICST.WICON2008.4929

trol messages are flooded from a gateway through the network in order to establish routes and provides loose time synchronization for reply messages. Each level in the tree determined by the path of the flooding message has a delay relative to the maximum depth of the tree (based on hop count) such that sensor nodes contend for the channel only with other nodes at the same depth. Figure 1 illustrates how a tree is formed over a topology. The spanning tree created by the downstream flooding message sets the routing tables for reply messages to the gateway. Since lower branches in the tree reply earlier, messages can be efficiently aggregated together as they are passed up the tree. By default, the aggregation phase of operation simply removes redundant messages however this can be easily enhanced to do more intelligent in network processing to reduce data. During the upstream period of data collection, the Low-Power-Listen (LPL) check rate (described in [2]) can be increased to allow for faster and more energy efficient collection since the network is expecting data. Routes can potentially change on each query making the network resilient to failures and able to support a high degree of mobility. Routes can optionally be reinforced over time to stop unnecessary re-routing. The flooding messages contain multiple control parameters setting attributes including: which nodes should reply, which sensor or data values should be transmitted, nonce counter values for CBC encryption, actuation control packets, network allocation vector (NAV) indicating when mobile communication is allowed. Control messages from the gateway can optimize the construction of the network spanning tree by applying various routing metrics or if need be set static routes.

While performing data collection, SAMPL operates in a push-pull cycle (see Figure 1) where queries are sent out to all nodes in the subnet and data is aggregate back to the gateway. In between requests, SAMPL operates in a Peer-to-Peer (P2P) mode. P2P requests could originate from mobile devices or infrastructure nodes in the form of asynchronous messages. Since much of the infrastructure is battery operated, network reservations are put in place to protect static nodes from battery depletion. Infrastructure nodes are allotted a fixed number of P2P transactions per period of time. This enforcement mechanism is configurable in the gateway's control packets. Mobile nodes that overhear packets from the infrastructure can identify the type of protocol, the version of the protocol and the next period of free network access time based upon packet header information. When the channel is clear, mobile nodes can then perform operations like pinging for lists of neighbors with corresponding RSSI values, or access nearby sensors. We also provide mobile nodes with access to read and write 4KB of EEPROM on all infrastructure nodes. This can be used to store various types of spatial information such as node GPS coordinates; messages for a friend that is running late; or conference room schedule information. By providing a general read and write interface, SAMPL leaves the usage of the data up to the application developers. This is extremely useful in scenarios such as maintenance, where the technician can store information such as the last service date, known issues, next scheduled service etc. on the device itself.

SAMPL has the following built-in packet types:

- **Transducer Packets** are used to request sensor values and to send actuation commands. These packets are variable length and include device specific type messages that allow requests to be heterogeneous.
- **Network Configuration Packets** are used to configure static networking parameters that are not configured during a gateway request. These currently include clear channel assessment thresholds, communication reservations, and debugging flags.
- **Runtime Statistics Packets** are used to report various properties at each node including; number of tx packets, number of rx packets, number of tx retries, number of rx failures, number of sensor samples, node uptime, processor deep sleep time and processor idle time. These parameters can be used to determine how much energy the device is consuming as compared to its energy budget.
- **Trace-Route Packets** are used to determine the route to the gateway through flooding. These packets record the MAC addresses as they are flooded across the network, to gather the required route information.
- **ACK/NACK Packets** are general replies that can be sent from any command that does not require return data.
- **EEPROM Message Packets** can be used to locally store messages at each node. This is similar to the data tags that can be stored on some RFID devices.
- **Neighborhood Packets** contain neighborhood information with associated link quality metrics. These lists are generated over time as nodes overhear neighbors.
- **Static Routing Packets** are used to configure static routes in the network that have precedence over the adaptive routes generated by the downstream routing messages. Routes have timeout values associated with them so that incorrectly configured nodes can eventually return to the default network state.

There is built-in support for CBC encryption, which is seeded of a nonce value transmitted as a part of the network configuration packets. The encryption is done in hardware by the cc2420 transceiver chip used in the FireFly node. Development efforts are underway for a public-key infrastructure setup for the establishment of encryption keys and enabling secure authentication with the network.

2.1 Runtime Visibility

A key feature of SAMPL is transparent support for network diagnostics. In Section 4 we describe a passive technique based on conditional probabilities of packet arrival for determining if particular nodes are being detrimental to the overall network. This approach does not expose information about how much energy the CPU and other tasks in the system are consuming. To support this information we provide an extensive set of debugging counters in each node that can be periodically requested through use of a runtime statistics packet. This packet contains a counter on each node for the number of packets transmitted, received, the number of transmit retries, the number of receive packet failures, the number of sensor sample requests, the node uptime, the processor deep-sleep time and the processor idle

Digital Object Identifier: 10.4108/ICST.WICON2008.4929
http://dx.doi.org/10.4108/ICST.WICON2008.4929

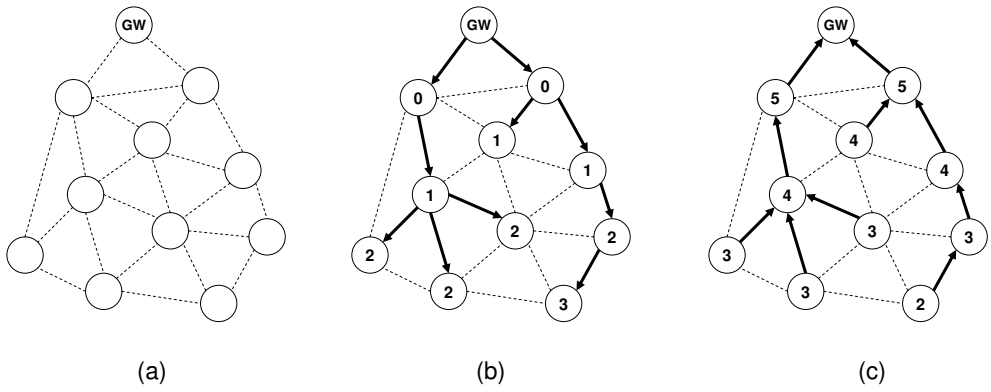

Figure 1: Retrieving sensor values using the SAMPL protocol. (a) example topology. (b) tree formed by the gateway as it sends a control message. This control message contains instructions on what operations the nodes should perform, which nodes should perform those operations as well as various parameters on how the tree should propagate and respond. The number inside each node indicates the level in the tree at which the message was received. The final figure (c) delay time before each node transmits based on the maximum hop count (in this case 5) and the delay per level. Notice that the nodes at the bottom of the tree transmit first allowing nodes higher up the tree to aggregate their data. During this collection mode of operation, the LPL sampling rate can be increased since each node is expecting data.

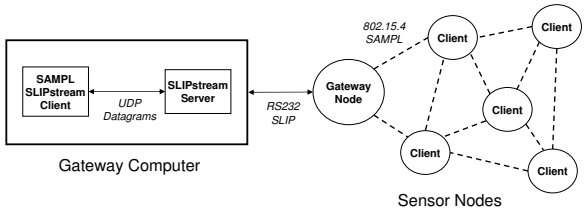

Figure 2: Location of various SAMPL protocol elements.

time. Runtime aggregation of such information leads to a highly visible network, whose health status is continuously monitored and archived over long durations of time.

2.2 Sensor Network to Server Interface

In order to interface desktop class applications with the sensor network, we introduce the SLIPstream protocol. SLIPstream is composed of three components: the SLIPstream server, the SLIPstream sensor node client and the SLIPstream UDP client (see Figure 2). The SLIPstream server translates UDP packets into serial datagrams and vice-versa. The SLIPstream server is designed to execute on an embedded Linux device or PC that is acting as a bridge between an IP network and the sensor network. UDP messages sent to the SLIPstream server are forwarded onto the sensor node using the Serial Line IP (SLIP) protocol. Unlike a simple serial to socket forwarder, the SLIP protocol allows for framing and checksumming of datagram packets going into and out of a sensor node. Since SLIP packets have special escape characters, this data can co-exist with normal debug messages that transmit as ASCII readable text. We provide simple transmit and receive functions for both the SLIPstream sensor node client as well as the SLIPstream UDP client.

The practical benefits of the SLIPstream infrastructure are realized in our gateway design. We have developed our own custom gateway (see Figure 4) device consisting of both a gumstix node with 802.11 interface and a 802.15.4 FireFly node, connected by a serial bus. The SLIPstream server runs on the gumstix node, and any server in the Internet can talk to the server using UDP packets over the WiFi interface. This enables an ultra-flexible interface to collect the aggregated data from the sub-network of sensors.

3. DEPLOYMENT

In this section we give a brief introduction to the Sensor Andrew project as well as the hardware platform and the operating system used to support SAMPL. We then discuss our debugging infrastructure and deployment approach.

3.1 Sensor Andrew

Sensor Andrew is a multi-disciplinary campus-wide scalable sensor network that is designed to host a wide range of sensing and low-power applications. The goals of Sensor Andrew are to support ubiquitous large-scale monitoring and control of infrastructure in a way that is extensible, easy to use, and provides security while maintaining privacy. Target applications currently being developed include infrastructure monitoring, first-responder support, quality of life for the disabled, water distribution systems monitoring and optimization, building power monitoring and control, social networking, and biometric sensors for campus security. A large component to these applications is an underlying wireless sensor network comprised of the Nano-RK real-time operating system running on the FireFly sensor networking platform.

3.2 Nano-RK

Nano-RK is a fully preemptive reservation-based real-time operating system (RTOS) with multi-hop networking sup-

Digital Object Identifier: 10.4108/ICST.WICON2008.4929
http://dx.doi.org/10.4108/ICST.WICON2008.4929

port for wireless sensor networks. It includes a light-weight embedded resource kernel (RK) with rich functionality and timing support capable of running on low-power micro-controllers. Nano-RK supports fixed-priority preemptive multitasking for ensuring that task deadlines are met, along with support for CPU, network, as well as, sensor and actuator reservations. Tasks can specify their resource demands and the operating system provides timely, guaranteed and controlled access to CPU cycles and network packets. Together these resources form virtual energy reservations that allows the OS to enforce system and task level energy budgets.

3.3 FireFly Hardware

The FireFly Sensor Networking Platform is a low-cost low-power hardware platform shown above. In order to better support real-time applications, the system is built around maintaining global time synchronization. The main Firefly board uses an Atmel ATmega1281 8-bit micro-controller with 8KB of RAM and 128KB of ROM along with ChipconâĂŹs CC2420 IEEE 802.15.4 standard-compliant radio transceiver for communication. The maximum packet size supported by 802.15.4 is 128 bytes and the maximum raw data rate is 250Kbps. The FireFly board supports various external peripherals such as a sensor expansion card, high voltage power monitoring and control board, and a firefly-hardware-clock-sync hardware clock synchronization module. The sensor expansion card provides light, temperature, audio, passive infrared motion, dual axis acceleration and voltage sensing. The high voltage power monitoring and control board allows for sensing of current draw as well as on/off actuation of 120VAC appliances. The base FireFly platform provides an SDIO port which can be used for large flash storage or as a universal interface to PC compatible peripherals.

In our current Sensor Andrew deployment, the FireFly nodes operate off of two D-cell sized batteries and communicate over multiple hops to a powered gateway that has access to the Internet. The sensor network is primarily designed to efficiently collect sensing data, however it also provides support for various mobile device interactions. We provide a generic communication interface allowing nodes to directly query infrastructure nodes as well as send messages to and from the Internet via the gateway. Communication reservations in Nano-RK provide a mobile node communication budget preventing mobile devices from draining more than their alloted system energy.

3.4 Debugging Infrastructure

Developing MAC protocols in sensor networks is challenging due to the physical separation of devices and the variations in wireless communication that hinders reproducibility. In this section we describe a testbed designed to provide a data back channel along with high precision local timing capabilities. Figure 3 shows one of the testbed nodes that consists of a gumstix based embedded Linux board, with a FireFly programming node and a FireFly sensor node. The embedded Linux box has an 802.11 interface that allows for remote programming and facilitates a high-speed debugging channel. A central server manages farming out firmware images as well as collecting and sorting debugging data. Each programming board provides a utility called TimeScope that allows for precise timing of GPIO pin toggling coming from the attached sensor node. By running the Network Time

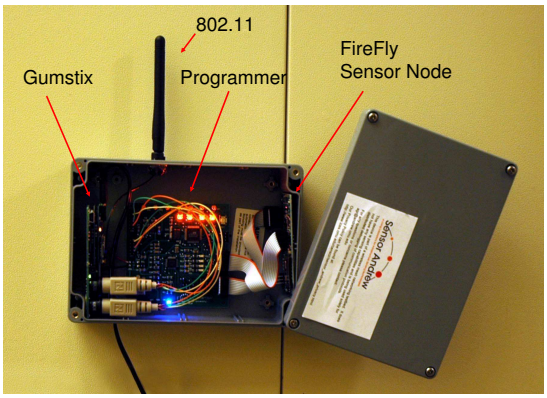

Figure 3: Testbed device with remote programming and debugging support. The note on the box summarizes the privacy policy of Sensor Andrew conforming to IRB regulations.

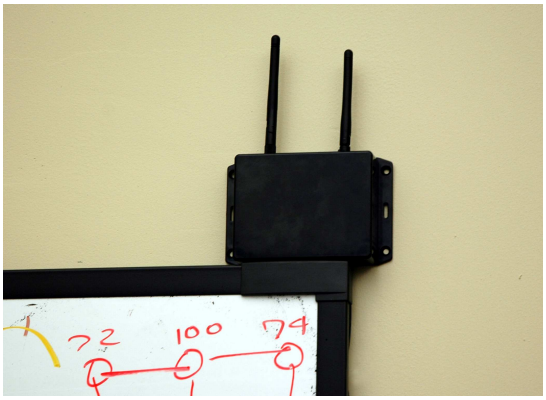

Figure 4: SAMPL gateway with an 802.15.4 and 802.11 interface.

Protocol [21] (NTP) on each Linux device and locally time-stamping debugging as well as timing messages we can build an account of what the network protocol is doing in a realistic topology.

The TimeScope utility built into all FireFly programming interface boards allows logic analyzer style debugging without interfering with serial output or bogging down the microcontroller with the large blocking times required for serial, SPI or I2C data transfers. Developers can assert and clear four different debugging GPIO pins on the FireFly board that are captured by the programmer board and can be used to generate timing waveforms. Figure 5 shows an example of timing output used to profile various sections of code. APIs on the FireFly node can also use this interface to send 4 bits of high speed debugging data back to your computer to track state transitions. The connections from the FireFly debugging board can even be linked to external logic devices that you wish to probe. TimeScope will give you in the worst case a 258 microsecond sampling period (3.875KHz) due to the time it takes to send the three bytes of payload over the UART. If signals do not occur back-to-back, then the factor limiting the resolution is the time-stamping period of 36

Digital Object Identifier: 10.4108/ICST.WICON2008.4929
http://dx.doi.org/10.4108/ICST.WICON2008.4929

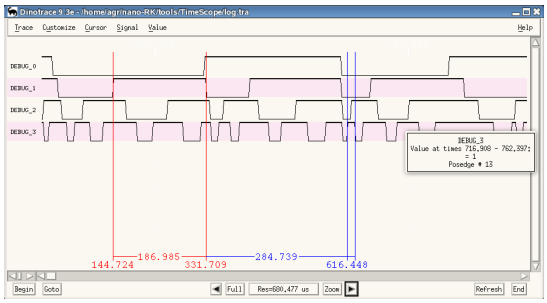

Figure 5: Timing output from TimeScope.

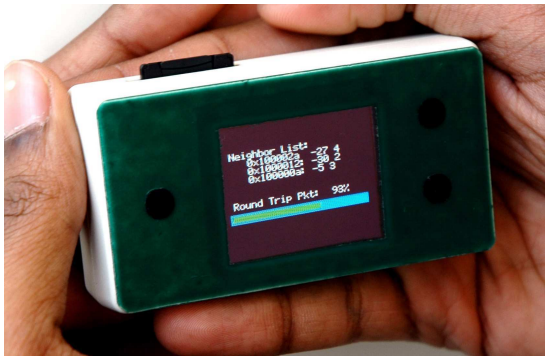

Figure 6: A mobile device, The-Radler, evaluating the current position for deploying an infrastructure node.

microseconds (27.7KHz).

One of the important motivations behind our project is to serve as a live testbed for sensor networking research. We therefore deployed a few nodes with support for reprogrammability. These nodes known as SAGs (Sensor Andrew Gateways) comprise of a (i) gumstix, (ii) a programmer, and (iii) a FireFly (see Figure 3). This subset of nodes can be reprogrammed on-the-fly, enabling us to evaluate the co-existence of other protocols, and testing future applications to be integrated into SAMPL.

3.5 Our Deployment Approach

The placement of nodes in a sensor network is important both for establishing reliable connectivity and to effectively sample the target environment. In many cases, node placement is governed by the application. For example if the node is supposed to monitor the vibration of a machine, it naturally needs to be physically place on the machine. In many cases, fortunately, nodes are simply forwarding data and/or can be placed practically any location within a large region of space. In this section, we describe strategies we use to effectively place nodes in the environment. We provide a simple mathematical framework to identify problematic regions of the network, that when adjusted, improve overall performance. In these examples, we assume the underlying sensor network MAC protocol is SAMPL. However, the same principles can easily be applied to other protocols.

When deploying nodes in a new location, our goal is to place the nodes covering as much area as possible while satisfying the following two requirements: (a) each node must have at least two disjoint and symmetric paths back to a gateway and (b) each link along this path should be of a sufficient link quality. We begin by placing a gateway in a central location with respect to our planned area of coverage. This helps to minimize the depth of the network which in turn reduces hop count and hence cumulative packet loss. Online probing of the network is required to maintain multiple disjoint paths. Figure 6 shows The-Radler a sensor node with three buttons and an LED screen. The deployment node sends round-trip `trace route` messages back to the gateway displaying the results on the screen. The trace route packets record node MAC addresses as it is flooded across the network. This makes it possible for receivers of multiple packets to determine if the route was disjoint or not. On the screen in Figure 6 we see an example showing the current position with three neighbors and their associated RSSI values. As an initial starting point, conservative RSSI thresholds averaged over time provide a reasonable indicator of link stability. The deployment node shows a color-coded bar graph indicating the success rate of packets reaching the gateway. This allows the deployment team to experiment with various test locations before they fix the node.

4. NETWORK ANALYSIS AND EVALUATION

Once a network has been deployed, we need to monitor how efficiently it is operating. One approach is to require active mechanisms so that the MAC protocol collects runtime statistics. This would include information such as neighbor lists and packet loss. For aggregation protocols, such as SAMPL, we can gain insights about the network over time using passive approaches. These techniques have the dual advantage of being both non-intrusive and highly economical from an energy perspective. In the following sub-section we look at what information can be gained simply by recording packet loss at the gateway.

4.1 Passive Analysis

Each time the network requests data, the lack of response from certain nodes provides information about weak communication links or node failures. Given the multi-hop and dynamic nature of routes in sensor networks, it can be hard to determine which nodes are most responsible for these packet losses. We present a scheme to identify clusters of related problem nodes that only requires keeping track of per-node packet loss at the gateway. By collecting correlated packet loss over time we can create a mapping of which nodes are most responsible for other nodes dropping packets. Based on the way we have deployed nodes, the sensor network should have enough redundancy that packet loss should not be dependent on any one particular node. Using the packet arrival data at the gateway, we generate a matrix that captures the inter-dependence of nodes throughout a subnet.

The packet loss matrix contains an NxN set of conditional probabilities relating how each node performs given the packet loss of all other nodes in the system. For each node in the matrix we record the count of packets dropped from node N_i as well as node N_j for each network request.

The total of packets N_i that are dropped in the same request as N_j divided by the total N_i gives us a sample probability that N_i drops packets given N_j dropped a packet. Using a two-proportion z-test assuming unequal variance we

Digital Object Identifier: 10.4108/ICST.WICON2008.4929
http://dx.doi.org/10.4108/ICST.WICON2008.4929

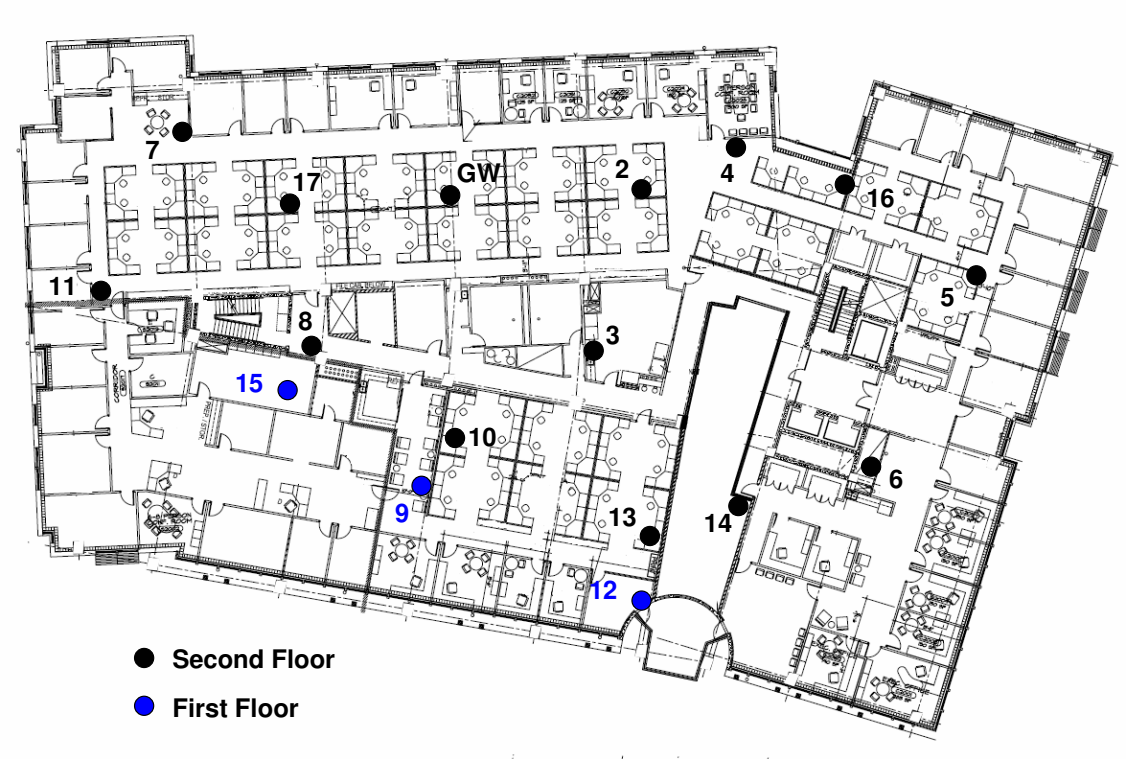

Figure 7: Firefly node placement in one of the Sensor Andrew building deployments.

can determine which nodes have packet loss that is statistically highly dependent on other nodes. This approach can automatically identify routes that are losing packets due to bottleneck nodes with poor links. Traversing the matrix can identify the set of nodes that most contributes to any particular nodes performance. The test is performed as follows:
Assumption:

P_i: probability that node i drops its packet

$P_{i|j}$: probability that node i drops packet given node j drops a packet
Hypothesis:

$$P_i - P_{i|j} = 0 \tag{1}$$

(i.e. packet loss is independent)
Next, we perform a Two-Proportion z-test with unequal variances on each node pair to determine packet loss independence:

$$z_{i,j} = \frac{(\hat{P_i} - \hat{P_{i|j}}) - (P_i - P_{i|j})}{\sqrt{\frac{\hat{P_i}(1-\hat{P_i})}{n_i} + \frac{\hat{P_{i|j}}(1-\hat{P_{i|j}})}{n_j}}} \tag{2}$$

$\hat{P_i}$ is the sampled packet loss for node N_i, $\hat{P_{i|j}}$ is the sampled packet loss for node N_i given packet loss in node N_j, n_i is the number of $\hat{P_i}$ samples and n_j is the number of $\hat{P_{i|j}}$ samples. $\hat{P_i}$ should be sampled randomly and independently of $\hat{P_{i|j}}$ since they are assumed to be independent. The resulting statistic for each z should be normally distributed. Since we are doing multiple tests across the dataset, we set a conservative z value threshold t, with t equal to an *alpha* of .05 divided by the number of nodes in the system N choose 2. Any values in the resulting z-test matrix that is above threshold t indicates that the node identified by that column has experience packet loss correlated with the row indexed node. In an ideal environment without correlated packet loss, the system will generate the identity matrix where nodes only experience packet loss from themselves. Since nodes forward data on behalf of other nodes, this is almost never the case. We will now show an example of how we used this method to monitor and optimize a deployment.

Figure 7 shows the layout of a deployment consisting of 17 sensor nodes with debugging facilities in one of our deployments at Carnegie Mellon University. The nodes were deployed using the strategy described in section 3 with the exception of nodes 16 and 17. These were added later based on information from our monitoring process. Figure 9 shows a histogram of packet loss for each node in the network based on over forty thousand packets collected every 30 seconds for two weeks. A client subscribed to data coming from the nodes continuously aggregates values for the packet loss matrix. Figure 8 shows the z-test results with values above a threshold of 3.5. Each column and row represents a node from 1 to 15. One can clearly see the diagonal of the matrix shows a high correlation since a node is always correlated with its own packet loss. Each non-zero value found down a column represents a node that is effectively failing to deliver packets in a manner that is causing a bottleneck. If a node is dropping packets, but that same packet is able to reach

Digital Object Identifier: 10.4108/ICST.WICON2008.4929
http://dx.doi.org/10.4108/ICST.WICON2008.4929

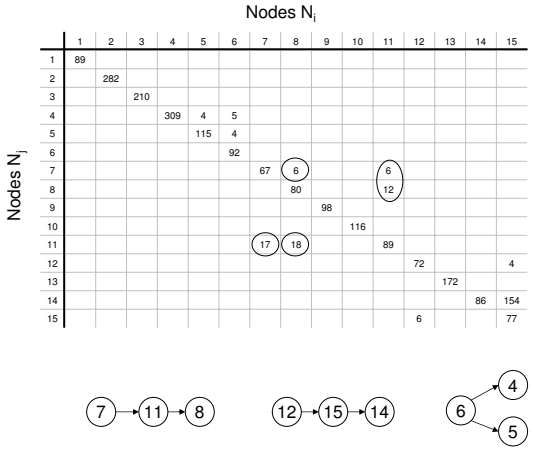

Figure 8: Thresholded z test packet loss matrix for the topology in Figure 7. Problem clusters with higher than normal node packet loss dependence are shown below the matrix. Values used to construct node 7's cluster are circled insider the matrix.

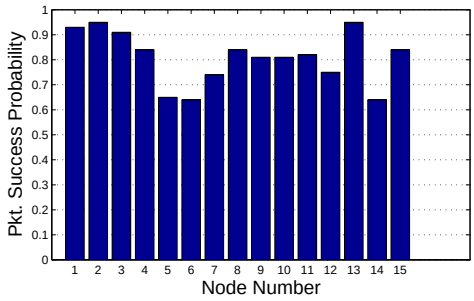

Figure 9: Packet success probability immediately after deployment.

the gateway through other means, this would not appear in the matrix.

4.2 Network Fortification

The packet loss matrix along with the map of the building shows examples of correlated packet loss which make sense given the network layout. For example, in column 6 we see that node 6's packets are largely correlated with drops in nodes 4 and 5. As can be seen on the map, nodes 4 and 5 are likely candidates to be forwarding node 6's traffic. The magnitude of the z value indicates the confidence of the correlation based on the sample size. Below the matrix in Figure 8 we see three clusters of problem nodes. These clusters are automatically generated by picking an initial starting point and then recursing over all non-zero node references in each column of the matrix.

These clusters indicate regions that need extra provisioning. We accomplish this by adding nodes 16 and 17 to the network. We also investigated node 12 which had been blocked by a barrier. Figure 10 shows the resulting histogram over the next two week period. The overall average packet success rate increased from 0.82 to 0.94. In a large

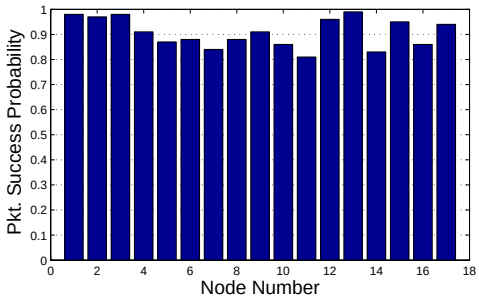

Figure 10: Packet success probability after optimizing the network. We see an overall increase in packet success probability from 0.82 to 0.94.

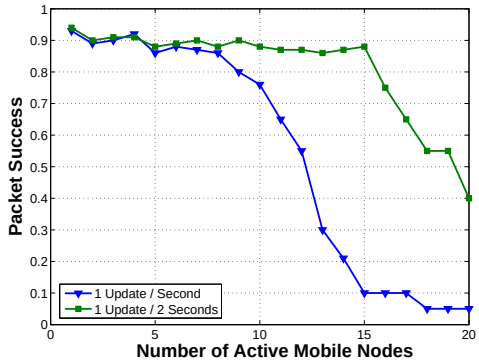

Figure 11: Limits on active mobile node density given a high demand on the infrastructure in a single collision domain.

scale deployment, it is important to have automated mechanisms for detecting problems and grouping problem nodes together.

4.3 Infrastructure Support for Mobile Devices

In order to evaluate the infrastructure support for mobile devices, we studied the packet success probabilities with increasing number of active mobile nodes (see Figure 11). It can be seen that the current implementation supports up to 15 mobile nodes with an update rate of once every second. As one would expect, increasing the mobile node update rate leads to more collisions, and hence a lower packet success probability as shown in the figure. These results are acceptable for most real-world scenarios, since the mobile devices are intelligent in nature, and use the infrastructure only sporadically. For instance, when the device detects that the user is moving around or that the environment is changing, it updates this new information to the infrastructure.

The design choices of SAMPL are inherently biased towards ensuring a higher reliability of packet delivery within the infrastructure than between the infrastructure and the mobile node. The rationale behind such a design is the fact that the battery-powered infrastructure nodes have to last for years, whereas the battery-powered mobile devices are typically charged every alternate night.

Digital Object Identifier: 10.4108/ICST.WICON2008.4929
http://dx.doi.org/10.4108/ICST.WICON2008.4929

5. CONCLUSIONS AND FUTURE WORK

In this paper, we have presented SAMPL a practical, dynamic, and extensible protocol for data aggregation and message passing in sensor networks. The protocol supports flexible-yet-efficient collection of network-wide sensor data, and provides infrastructure support for mobile devices. As of October 2008, the infrastructure is in use for well over seven months, and continues to be actively used as a research testbed. As a part of the whole system infrastructure, we have developed a wide range of debugging tools for continuously monitoring and collecting information about the health of the network. In this paper, we have described some of these tools like TimeScope and SAG in detail. In order to efficiently archive data from the sensor network, we also developed the SLIPstream service to enable communication over UDP. One of the key issues we faced during our deployment phase is that of ensuring sufficient coverage and redundancy. Our mobile deployment device with its graphical user interface, was a significant aid in overcoming these challenges. Finally, we have designed efficient statistical tests to understand network-wide packet-loss patterns, and demonstrated how they can be applied to improve network reliability. Our system has achieved an average sustainable packet success rate of 94% across the entire network.

Future work involves scaling to thousands of nodes with multiple sub-networks, and integrating with numerous pre-existing legacy protocols.

6. REFERENCES

[1] A. Eswaran, A. Rowe and R. Rajkumar. Nano-RK: an Energy-aware Resource-centric RTOS for Sensor Networks. *IEEE Real-Time Systems Symposium*, 2005.

[2] J. Polastre, J. Hill and D. Culler. Versatile low power media access for wireless sensor networks. *SenSys*, November 2005.

[3] Mainwaring A., Polastre J., Szewczyk R., Culler D., Anderson J. Wireless Sensor Networks for Habitat Monitoring. *ACM International Workshop on Wireless Sensor Networks and Applications*, 2002.

[4] Philo Juang, Hidekazu Oki, Yong Wang, Margaret Martonosi, Li-Shiuan Peh, and Daniel Rubenstein. Energy-Efficient Computing for Wildlife Tracking: Design Tradeoffs and Early Experiences with ZebraNet. *ASPLOS-X*, 2002.

[5] W. Ye, J. Heidemann and D. Estrin. An energy-efficient mac protocol for wireless sensor networks. *INFOCOM*, June 2002.

[6] Rowe A., Mangharam R., and Rajkumar R. RT-Link: A Time-Synchronized Link Protocol for Energy-Constrained Multi-hop Wireless Networks. *SECON*, 2006.

[7] V. Rajendran, K. Obraczka and J. J. Garcia-Luna-Aceves. Energy-efficient, collision-free medium access control for wireless sensor networks. *Sensys*, 2003.

[8] T. Dam and K. Langendoen. An adaptive energy-efficient mac protocol for wireless sensor networks. *SenSys*, November 2003.

[9] A. El-Hoiydi and J. Decotignie. Wisemac: An ultra low power mac protocol for the downlink of infrastructure wireless sensor networks. *ISCC*, 2004.

[10] L.F.W. van Hoesel and P.J.M. Havinga. A lightweight medium access protocol for wireless sensor networks. *INSS*, 2004.

[11] C. Guo and L. C. Zhong and J. Rabaey. Low power distributed mac for ad hoc sensor radio networks. *Globecom*, 2001.

[12] Culler D., Dutta P., Tien Ee C., Fonseca R., Hui J., Levis P., Polastre J., Shenker S., Stoica I., Tolle G., Zhao J. Towards a Sensor Network Architecture: Lowering the Waistline. *Proceedings of the Tenth Workshop on Hot Topics in Operating Systems (HotOS X)*, 2005.

[13] Heidemann J., Silva F., Intanagonwiwat C., Govindan R., Estrin D., Ganesan D. Building Efficient Wireless Sensor Networks with Low-Level Naming. *SOSP*, 2001.

[14] S. Madden, M. J. Franklin, J. M. Hellerstein and W. Hong. TAG: A Tiny AGgregation Service for Ad-Hoc Sensor Networks. *Operating Systems Design and Implementation (OSDI)*, 2002.

[15] Maurer, U., Rowe, A., Smailagic, A., Siewiorek, D. eWatch: A Wearable Sensor and Notification Platform. *IEEE Workshop on Wearable and Implantable Body Sensor Networks*, 2006.

[16] Krumm, J., Williams, L., Smith G. SmartMoveX on a Graph - An Inexpensive Active Badge Tracker. *UbiComp*, 2002.

[17] Laibowitz M., Gips J., Aylward R., Pentland A., Paradiso J. A Sensor Network for Social Dynamics. *International Conference on Information Processing in Sensor Networks (IPSN)*, 2006.

[18] Want, A., Jones, A., Hopper, A. A New Location Technique for the Active Office. *IEEE Personal Comm.*, 1997.

[19] J. Zhao and R. Govindan. Understanding packet delivery performance in dense wireless sensor networks. *Proc. ACM Sensys*, 2003.

[20] Randolph D. Nelson and Leonard Kleinrock. Maximum probability of successful transmission in a random planar packet radio network. *INFOCOM*, pages 365–370, 1983.

[21] Mills, D.L. Internet time synchronization: the Network Time Protocol. *IEEE Trans. Communications COM*, 1989.

Digital Object Identifier: 10.4108/ICST.WICON2008.4929
http://dx.doi.org/10.4108/ICST.WICON2008.4929