Chapter #

SCALABLE, SELF-ORGANIZING TECHNOLOGY FOR SENSOR NETWORKS

Kenneth P. Birman, Saikat Guha, Rohan Murty

Dept. of Computer Science, Cornell University; Ithaca, New York 14853

Abstract: Sensor networks will often need to organize themselves automatically and adapt to changing environmental conditions, failures, intermittent connectivity, and in response to power considerations. We review a series of technologies that we find interesting both because they solve fundamental problems seen in these settings, and also because they appear to be instances of a broader class of solutions responsive to these objectives.

Key words: Embedded computing, sensors, distributed monitoring, routing, network position information.

1. INTRODUCTION

The emergence of a new generation of technologies for pervasive computing and networking is challenging basic assumptions about how networked applications should behave. Wired systems typically ignore power and location considerations and operate "in the dark" with respect to overall system configuration, current operating modes or detected environmental properties, and positions of devices both in absolute and logical terms. These kinds of assumptions represent serious constraints and lead to sub-optimal solutions in embedded or pervasive computing systems.

At Cornell, we and other researchers are working to develop platform technologies responsive to these and related considerations. This paper reports on three representative examples, which we offer with two goals in mind. First, each of these technologies reflects a mixture of properties and algorithmic features matching the special requirements seen in pervasive computing settings. Second, we believe that the underlying methodologies reflected in the three technologies are interesting in themselves, because they point to broader opportunities for future study.

At the time of this writing, the technologies are not integrated into a single platform, but doing so is an eventual goal. Indeed, we believe that over time, researchers will conclude that pervasive computing systems demand a completely new kind of infrastructure, built using components such as the ones we present here.

Specific properties of importance include the following. First, our solutions are self-organizing, a crucial property in many emerging applications. They are strongly self-stabilizing, converging rapidly to a desired structure and repairing themselves rapidly after disruption. They lend themselves to theoretical modeling and analysis, lending themselves to both pencil-and-paper study and to simulation. Significantly (and unlike many distributed systems technologies), the analyses so obtained hold up well in practice; as we'll see below, this is because our protocols are so overwhelmingly convergent. Moreover, they are robust to perturbation, a property that may be extremely important in the relatively turbulent world in which many sensor applications will need to operate. Interestingly, each solution consists of a relatively simple protocol run in parallel by the components of the system, and the desired global outcome "emerges" rapidly through the interaction of a component with its neighbors. We conjecture that a rich class of solutions having these properties awaits discovery by future researchers.

The three services on which we focus here are (1) *Astrolabe*, a system for distributed state monitoring, application management, and data mining constructed using a novel peer-to-peer protocol that offers unique scalability, low load, and rapid convergence; (2) *Tycho*, a location-aware event localization system for sensor networks, and (3) *Sextant*, a system for discovering sensor locations using software methods that is highly accurate, energy-efficient and scalable. Each is really an instance from a broader class of related solutions, and is interesting both in its own terms, but also as exemplars of these broader classes.

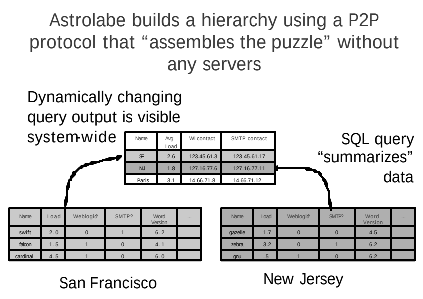
2. ASTROLABE

The Astrolabe system is best understood as a relational database built using a peer-to-peer protocol running between the applications or computers on which Astrolabe is installed. Like any relational database, the fundamental building block employed by Astrolabe is a tuple (a row of data items) into which values can be stored. For simplicity in this paper, we'll focus on the case where each tuple contains information associated with some computer. The technology is quite general, however, and can be configured with a tuple per application, or even with a tuple for each instance of some type of file or database. For the purposes of this paper, Astrolabe would be used to capture and track information associated with sensors in a network of sensors. For reasons of brevity, this section (and those that follow) limits itself to a brief overview; additional detail can be found in [1] and [2].

The data stored into Astrolabe can be drawn from any of a number of sources. Small sensors would export data "directly" but a larger, more comprehensive computing node could export more or less any kind of data that can be encoded efficiently. This includes information in the management information base (MIB), fields extracted directly from a file, database, spreadsheet, or information fetched from a user-supplied method associated with some application program. Astrolabe is also flexible about data types, supporting the usual basic types but also allowing the application to supply arbitrary information encoded with XML. The only requirement is that the total size of the tuple be no more than a few k-bytes; much larger objects should be handled outside the core Astrolabe framework.

The specific data that should be pulled into Astrolabe is specified in a configuration certificate. Should the needs of the user change, the configuration certificate can be modified and, within a few seconds, Astrolabe will reconfigure itself accordingly. This action is, however, restricted by a security policy, details of which are described in [1] and [2].

Astrolabe groups small sets of tuples into a hierarchy of relational tables. A "leaf" table consists of perhaps 30 to 60 tuples (we could scale up to hundreds but not thousands of types in a single table) containing data from sources physically close to one-another in the network. This grouping (a database administrator would recognize it as a form of schema) can often be created automatically, using latency and network addresses to identify nearby machines (the location information could, for example, be obtained using the method we present in Section 4 of this paper).


Astrolabe builds a hierarchy using a P2P protocol that "assembles the puzzle" without any servers
Dynamically changing query output is visible system-wide
SQL query "summarizes" data
San Francisco
New Jersey

Name	Avg Load	WLcontact	SMTP contact
SF	2.6	123.45.61.3	123.45.61.17
NJ	1.8	127.16.77.6	127.16.77.11
Paris	3.1	14.66.71.8	14.66.71.12

Name	Load	Weblogic?	SMTP?	Word Version	...
swift	2.0	0	1	6.2	
falcon	1.5	1	0	4.1	
cardinal	4.5	1	0	6.0	

Name	Load	Weblogic?	SMTP?	Word Version	...
gazelle	1.7	0	0	4.5	
zebra	3.2	0	1	6.2	
gnu	.5	1	0	6.2	

Figure #-1. Example of an Astrolabe representation of data extracted from a set of sensors.

The data collected by Astrolabe evolves as the underlying information sources report updates, hence the system constructs a continuously changing database using information that actually resides on the participating computers. Figure #-1 illustrates this: we see a collection of small database relations, each tuple corresponding to one machine, and each relation collecting tuples associated with some set of nearby machines. In this figure, the data stored within the tuple includes the name of the machine, its current load, an indication of whether or not various servers are running on it, and the "version" for some application. Keep in mind that this selection of data is completely determined by the configuration certificate. In principle, any data available on the machine or in any application running on the machine can be exported. In particular, spreadsheets and databases can easily be configured to export data to Astrolabe.

The same interfaces which enable us to fetch data so easily also make it easy for applications to use Astrolabe. Most commonly, an application would access the Astrolabe relations just as it might access any other table, database or spreadsheet. As updates occur, the application receives a form of event notifying it that the table should be rescanned. Thus, with little or no specialized programming, data from Astrolabe data could be « dragged » into a local database, spreadsheet, or even onto a web page. As the data changes, the associated application will receive refresh events.

Astrolabe is intended for use in very large networks, hence this form of direct access to local data cannot be used for the full dataset: while the system does capture data throughout the network, the amount of information would be unwieldy and the frequency of updates excessive. Accordingly, although Astrolabe does provide an interface whereby a remote region's data can be accessed, the normal way of monitoring remote data is through aggregation queries.

An aggregation query is, as the name suggests, just an SQL query which operates on these leaf relations, extracting a single summary tuple from each which reflects the globally significant information within the region. Sets of summary tuples are concatenated by Astrolabe to form summary relations (again, the size is typically 30 to 60 tuples each), and if the size of the system is large enough so that there will be several summary relations, this process is repeated at the next level up, and so forth. Astrolabe is thus a hierarchical relational database, and this is also visible in Figure #-1, where the summaries of the various regions appear as rows in the root relation. Each of the summaries is updated, in real-time, as the leaf data from which it was formed changes. Even in networks with thousands or millions of instrumented machines, updates are visible system-wide within a few tens of seconds. Since sensor networks may be very large, this scalability is likely to be important.

A computer using Astrolabe will, in general, keep a local copy of the data for its own region and aggregation (summary) data for region above it on the path to the root of this hierarchy. As just explained, the system maintains the abstraction of a hierarchical relational database. Physically, however, this hierarchy is an illusion, constructed using a peer-to-peer protocol, somewhat like a jig-saw puzzle in which each computer has ownership of one piece and read-only replicas of a few others. Our protocols permit the system to assemble the puzzle as a whole when needed. Thus, while the user thinks of Astrolabe as a somewhat constrained but rather general database, accessed using conventional programmer APIs and development tools, this abstraction is actually an illusion, created on the fly. In particular, the memory needed to run the system is very small, even in a network that may be very large.

The peer-to-peer protocol used for this purpose is, to first approximation, easily described. Each Astrolabe system keeps track of the other machines in its zone, and of a subset of contact machines in other zones. This subset is selected in a pseudo-random manner from the full membership of the system (again, a peer-to-peer mechanism is used to track approximate membership ; for simplicity of exposition we omit any details here). At some fixed frequency, typically every 2 to 5 seconds, each participating machine sends a concise state description to a randomly selected destination within this set of neighbors and remote contacts. The state description is very compact and lists versions of objects available from the sender. We call such a message a « gossip » event. Unless an object is very small, the gossip event will not contain the data associated with it.

Upon receiving a gossip message, an Astrolabe system is in a position to identify information which may be stale at the sender's machine (because timestamps are out of date) or that may be more current at the sender than on its own system. We say may because time elapses while messages traverse the network, hence no machine actually has current information about any other. Our protocols are purely asynchronous: when sending a message, the sender does not pause to wait for it to be received and, indeed, the protocol makes no effort to ensure that gossip gets to its destinations.

If a receiver of a gossip message discovers that it has data missing at the sender machine, a copy of that data is sent back to the sender. We call this a push event. Conversely, if the sender has data lacking at the receiver, a pull event occurs: a message is sent requesting a copy of the data in question. Again, these actions are entirely asynchronous; the idea is that they will usually be successful, but if not (e.g. if a message is lost in the network, received very late, or if some other kind of failure occurs), the same information will probably be obtained from some other source later.

One can see that through exchanges of gossip messages and data, information should propagate within a network over an exponentially increasing number of randomly selected paths among the participants. That is, if a machine updates its own row, after one round of gossip, the update will probably be found at two machines. After two rounds, the update will probably be at four machines, etc. In general, updates propagate in log of the system size – seconds or tens of seconds in our implementation. In practice, we configure Astrolabe to gossip rapidly within each zone (to take advantage of the presumably low latency) and less frequently between zones (to avoid overloading bottlenecks such as firewalls or shared network links). The effect of these steps is to ensure that the communication load on each machine using Astrolabe and also each communication link involved is bounded and independent of network size.

We've said that Astrolabe gossips about objects. In our work, a tuple is an object, but because of the hierarchy used by Astrolabe, a tuple would only be of interest to a receiver in the same region as the sender. In general, Astrolabe gossips about information of shared interest to the sender and receiver. This could include tuples in the regional database, but also aggregation results for aggregation zones that are ancestors of both the sender and receiver.

After a round of gossip or an update to its own tuple, Astrolabe recomputes any aggregation queries affected by the update. It then informs any local readers of the Astrolabe objects in question that their values have changed, and the associated application rereads the object and refreshes its state accordingly. The change would be expected to reach the server within a delay logarithmic in the size of the network, and proportional to the gossip rate. Using a 2-second gossip rate, an update would thus reach all members in a system of 10,000 computers in roughly 25 seconds. Of course, the gossip rate can be tuned to make the system run faster, or slower, depending on the importance of rapid responses and the available bandwidth.

Astrolabe was originally developed for use in very large-scale wired environments, but has several features well-matched to sensor networks and other embedded applications. First, most communication occurs between a components and nearby peers. In wireless ad-hoc routed networks, this is important because sending a message to a very remote component consumes power not just on the sender and receiver, but also on intermediary nodes involved in routing the packet. In a reasonably dense sensor network, most Astrolabe communication will occur between sensors near to one-another, with only aggregation information being transmitted over long distances.

Astrolabe doesn't rely on any single or even "primary" route between components that share information through its database; instead, within any zone data travels over all possible paths within that zone. This is important because it makes the protocol extremely robust to routing disturbances or transient communication problems. Astrolabe will report events within roughly the same amount of time even if serious disruption is occurring and the system repairs itself rapidly after failure or other stresses.

Finally, Astrolabe can be made to configure itself entirely automatically, using proximity within the network to define zones. In the sections that follow we'll see other uses of location information as an input to system configuration algorithms; we believe the idea is one that merits further study and broader use.

Astrolabe is just one of several technologies we've constructed using this methodology. Others relevant to pervasive computing include Bimodal Multicast [3], a scalable protocol that uses peer-to-peer epidemic protocols to achieve very high reliability at rather low cost, and Kelips [4], a novel distributed indexing mechanism (a "DHT" in the current peer-to-peer vocabulary). Kelips can find information for the cost of a single RPC even in a massive network. All of these solutions share strong similarities: inexpensive gossip-based protocols that converge because they mimic the propagation of an epidemic, constant background overheads (on component nodes and links), a preference for local communication, and very robust behavior even under stress. Moreover, precisely because of their overwhelmingly rapid convergence, even simplified theoretical models and analysis tend to be quite robust, yielding predictions that are later confirmed experimentally. Finally, all of these mechanisms have very simple implementations, small memory footprints, and use relatively low bandwidth.

Work still remains to be done: none of our protocols is able to deal with scheduled sleep periods or other power conservation and scheduling considerations. Nonetheless, we believe that they represent exciting starting points.

3. INTERACTIONS BETWEEN POWER AWARE COMPONENTS

A typical sensor tends to be very small in size. Though this serves as an advantage, it also limits the sensor's capabilities in terms of energy and data storage, processing power, and communication capabilities. As sensors mature, we believe that they will become faster, cheaper and be able to collect and store more data and transmit it farther; however, their energy capacity will not increase at a similar rate. There have been advances in the use of wireless power, and renewable sources of energy in sensors, but the technologies are yet to emerge from their infancy. In such light we believe that application and software level power conservation can help balance the energy budget enabling sensor networks to last longer than previously imagined.

We outline below some characteristics of power aware components for sensor-networks that aim to lengthen network lifetime.

- **Hardware re-use**: When possible, existing hardware should be used by multiple components instead of power-consuming single-purpose hardware.
- **Uniform energy dissipation**: Computational and communication load of a node should be proportional to the energy available to it.
- **Low duty-cycle**: Components should periodically allow some nodes to operate in power-saving mode.
- **Reactive protocols**: Reactive protocols or proactive protocols with localized effect should be used to minimize communication costs.
- **Configurable trade-off**: Network administrators should be able to trade-off system lifetime with system latency and effectiveness.

Research has shown communication to be far more expensive than computation for sensor networks [5]. Additionally, unreliable radio communication links further aggravate this problem. In Tycho [9], an event localization system for sensor-networks, we explore the case of a power-aware component where nodes send data to a single controller. In Tycho, a query is injected into the network and upon detection of an event, sensor nodes send the detected information to the controller. Therefore it is desirable to optimize communication links from multiple sources of data to a single destination (controller).

In the operation of a typical sensor network, various components such as the application, routing, location discovery etc. interact with one another. Research in this area has focused on minimizing energy consumption of each of these components when executing independently and when interacting with one another. In this section, we focus on minimizing the energy consumption through increased interactions between the application and the routing layers.

Consider a large sensor network in which nodes can either detect an event or be queried based on certain parameters. A simple approach to reporting results when an event occurs or when a query is passed on to the network, is to flood the controller with messages from each individual sensor node. While this is a simple enough mechanism to code into sensors, it results in inefficient usage of energy, and can fail to report accurate results when a sensor is not capable of communicating directly with the controller. Further, this approach does not scale well with an increase in the number of nodes in the network.

As a result, significant research effort in this area aims to improve the longevity of sensor nodes by optimizing for the sensor's power usage, minimizing communication and the size of messages transferred, without compromising on the functionality of the sensors. This is commonly achieved by installing a routing fabric in the network and then aggregating data along the routing path. An optimal routing protocol tends to either minimize the number of nodes involved in routing or minimize the distance each message is transmitted between adjacent nodes in the network. Energy can be further conserved by aggregating along the path to the controller. This approach is optimal in terms of the total energy consumed by the various nodes to either transmit or receive messages. Further, aggregation has the potential to reduce the number of messages transmitted between nodes as well as the size of the messages.

Various simulations and physical experiments have shown previously proposed protocols to have extended the life of a sensor network well beyond that of direct communication or the flooding approach previously described.

Routing protocols tend to expose functionality through limited interactions with the application. The first of such functionality is the decision regarding which nodes can be used in low duty cycle modes. A large fraction of energy savings are achieved by enabling a significant fraction of the nodes in the network to shift to a low duty cycle mode. While various routing protocols achieve low duty cycle nodes by putting various nodes in the network to sleep, other routing protocols do not achieve low duty cycle nodes since they require all nodes in the network to be alive during routing. A common scheme among routing protocols that support low duty cycle nodes is to interact with the application when deciding on the subset of nodes that are to go to sleep. Secondly, routing schemes decide on a path in the network between two nodes which are received as input from the application layer via APIs exposed by the routing layer.

Based on ongoing research in sensor networks at Cornell, we propose a two fold extension of the functionality of routing protocols and APIs exposed by them to the applications running on top. First, we propose an interaction between the routing layer and the application layer in which the application is able to decide on a specific subset of nodes in the network that should be involved in routing. The routing layer should then be able to construct paths between the specified set of nodes (multiple sources) and the controller by minimizing the additional nodes used. This should be achieved while preserving the energy efficiency of the routing protocol. Additionally, based on the specified subset of nodes, the routing layer should be capable of setting up these paths dynamically.

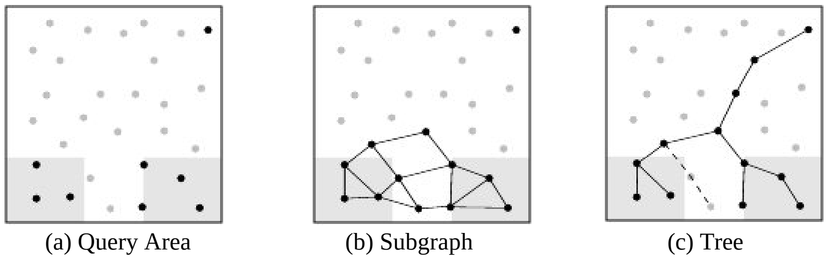


Figure #-2. Routing tree formation.

Consider the example illustrated in Figure #-2. A sensor network is setup such that the controller is located at the top right hand corner. The controller is interested in detecting events in the shaded region (query area) as shown in 2(a). Using techniques proposed in [6], the nodes that lie in the shaded region are determined. The sub-graph in Fig 2(b) represents the subset of nodes that lie in the shaded region and therefore are required to remain awake in order to be able to sense an event and to take part in routing. When forming a routing path from these nodes to the controller node, the number of additional nodes (that lie in the unshaded area) used are minimized as shown in 2(c). The remaining nodes in the network can now be shifted to a low duty cycle mode. In this example, the application running at the controller is required to interact with the routing layer and decide on the subset of nodes that are required to take part in routing. The routing layer should be flexible enough to dynamically form the routing paths based on these constraints. From preliminary results on our research, we have found that this approach leads to a significant amount of energy savings when compared to other energy efficient routing protocols described in [7, 8]. This particular extension has been included in the Tycho system currently being developed at Cornell. Additionally, it is desirable for sensors in the network to uniformly dissipate energy since this guarantees that no single sensor will drain its power prematurely thus possibly disconnecting fractions of the network. The routing layer should bear this responsibility, and it can achieve this by varying the subset of nodes that are sent to sleep during each subsequent sleep cycle.

Various routing protocols to reduce latency in event detection and data aggregation have been proposed. However, it is our belief that if the application is able to negotiate the maximum permissible latency with the routing layer, this will permit a greater degree of flexibility to the routing layer when making decision regarding which nodes should be put into a low duty cycle mode. This is primarily because not all sensor network applications require a low latency network. If an application is able to tolerate high latency in receiving data from an event, the routing layer is then given the freedom to put a large fraction of nodes in the network to sleep. Therefore, when an event is detected by a node, if nodes along the routing path are asleep, data can be cached and sensors can wait till every other sensor on the routing path is awake and then transmits the information to the controller. As a result, the energy saved is a function of the permissible latency. This is the second extension we propose to the interactions between the application and the routing layers.

The interactions described in this section yield more flexibility to the applications and require the various layers to work together cohesively with the goal of saving energy.

4. POSITION INFORMATION IN SENSOR NETWORKS

Sensor networks by definition sense their surroundings and cannot afford to operate "in the dark" with respect to their position in the field. Position information is necessary for tagging sensor readings [11,12], geographic routing [10] and caching schemes, clustering and group formation schemes useful in Astrolabe [1], and even addressing the sensors in certain applications [13]. To enable such a wide range of sensor-network applications the community is in search for the perfect sensor-network networking stack and we believe that position services will figure prominently in such a stack. Research in this area has traditionally focused on a very constrictive API for location services, one which consists of a single function that returns a best-estimate point-location for a sensor. This approach is adequate for small-scale static networks where sensor locations can be statically programmed in at deployment time but does not scale well to large or dynamic networks. At Cornell, ongoing research aims to provide a flexible and scalable location-discovery component for the sensor-network networking stack.

As mentioned in the Sextant paper [6] an ideal location discovery protocol would have the following properties:

- **Cheap**: Location discovery should be cheap and consume little power, with minimal dependence on infrastructure in the environment and dedicated hardware on each node.
- **Accurate**: Location discovery should achieve high accuracy. The degree of accuracy should be tunable by the network administrator.
- **Scalable**: The protocol should scale well with increasing number of nodes. Communication load on a node should be independent of the total area of network coverage and the total number of nodes and computation should be distributed evenly.
- **Heterogeneous**: The protocol should support heterogeneous networks where nodes have differing capabilities, such as varying transmission power levels, antenna arrays for determining angle of arrival, configurable angle of transmission and signal strength measurement hardware for relative position estimation.
- **Easy to deploy**: Finally, the protocol should be practical and easy to deploy. Assumptions made in calculating locations should hold in the field.

Sextant, designed with the above goals in mind, necessitates a new API for location services that allows it to return high-fidelity location data and facilitates two-way interactions between Sextant and the applications using it.

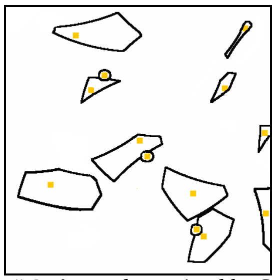


Figure #-3. Areas determined by Sextant.

A critical issue in location discovery is the representation of a node's position. One approach is to keep and update only a single point estimate for a node. While this approach requires little state, it introduces errors that may compound at the location discovery level as well as at the application level. Sextant, instead, explicitly tracks and refines over time the area a sensor can be located within. To maximize accuracy and minimize storage and communication requirements, it uses Bezier curves to efficiently represent the areas, which need not be convex or even simply connected. Figure #-3 depicts nodes along with their Sextant areas. While in its current state Sextant gives applications the guarantee that the sensor resides within the area determined, a simple extension can annotate the returned area with a probability distribution which represents the relative confidence of the system in the node's precise position.

A second issue is the collection of location information. The economic and energy cost of using dedicated positioning hardware, like GPS receivers, at each node is prohibitively high. Instead Sextant infers location information by creating a system of equations, the solution for which gives the area within which each node may be located, and then solves this system in a distributed fashion. Sextant uses the communication hardware already present at each node as a primary source of geographic constraints that it translates into the system of equations. In addition, it can generate additional constraints from other sources including event-sensors and antenna arrays when available. Sextant uses a very small number of landmark nodes that are aware of their own locations, either by static encoding or the use of GPS, to arrive at its solution.

As a power-aware component, Sextant adheres to the guidelines discussed in the previous section by reducing its dependence on power-consuming dedicated positioning hardware like GPS receivers and depending instead on MAC level information gleaned from the already present communication hardware. In addition, Sextant converges quickly in static networks obviating the need for constant Sextant-traffic and in highly dynamic networks it limits itself to localized proactive traffic and evenly distributes the processing and communication load leading to uniform energy dissipation in the network.

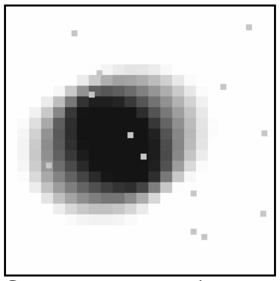


Figure #-4. Tycho using Sextant areas to assign event detection probabilities.

Of the applications that use the Sextant API, legacy applications can query Sextant for a point-estimate that they are well suited to deal with, while applications that that can use the extra information available can do so to minimize their own error. This later approach is evaluated in the Tycho paper [9] where the system, built on top of Sextant, weighs data from different sensors based on the confidence of the sensor's position to produce a probability distribution of an event's location. Figure #-4 illustrates a node's confidence in the event's location given the Sextant area it lies within. The likelihood of the event taking place in an area is represented using different shades with lighter shades representing low probabilities and darker shades representing high probabilities. Tycho has been shown to be more accurate than traditional triangulation schemes used previously that locate an event to a point-location. In addition, the new API allows Tycho to provide Sextant with additional geographic constraints that it gleans from the already present sensor hardware. This serves to enhance Sextant's location estimates and in turn iteratively increases Tycho's accuracy.

5. CONCLUSIONS

Our paper reviewed three technologies matched to the unique needs of pervasive computing environments. Although the components have yet to be integrated into a single platform, doing so is an eventual objective of our effort.

In fact, we believe that the ideas underlying the solutions we present here would also be useful in wired systems. For decades, developers have constructed wired network applications under the assumption that the less each application component "knows" about the network, or about the states of peer components, the better. This sort of thinking is reflected in the prevailing application development models and platforms: client-server systems in their varied forms, Web applications, and most recently the Web Services architecture. One can trace the underlying mindset to the end-to-end philosophy, which can be interpreted as arguing for black-box networks and application designs in which each component is on its own.

It may be time to explore a countervailing view, better matched to the properties of pervasive computing and embedded sensor applications. This view recognizes that the topology of a network, the properties of the components, their positions in the real world and relative to one-another and the constraints under which they operate may have implications for the behavior of other components. Such thinking argues for system services that make it easy for components to share their states and to exploit the information they obtain from one-another to achieve global objectives that would otherwise be unrealizable.

The end-to-end philosophy served us well in developing wired applications, but a new paradigm of sensitivity to system and network state may be needed in response to the unique needs of these new kinds of systems.

6. REFERENCES

[1] Astrolabe: A Robust and Scalable Technology for Distributed System Monitoring, Management, and Data Mining. Robbert van Renesse, Kenneth Birman and Werner Vogels. *ACM Transactions on Computer Systems,* May 2003, Vol.21, No. 2, pp 164-206.

[2] Scalable Data Fusion Using Astrolabe. Ken Birman, Robbert van Renesse and Werner Vogels. In proceedings of the Fifth International Conference on Information Fusion 2002 (IF 2002), July 2002.

[3] Bimodal Multicast. Kenneth P. Birman, Mark Hayden, Oznur Ozkasap, Zhen Xiao, Mihai Budiu and Yaron Minsky. ACM Transactions on Computer Systems, Vol. 17, No. 2, pp 41-88, May, 1999.

[4] Kelips: Building an Efficient and Stable P2P DHT Through Increased Memory and Background Overhead. Indranil Gupta, Ken Birman, Prakash Linga, Al Demers and Robbert van Renesse. Submitted to: 2nd International Workshop on Peer-to-Peer Systems (IPTPS '03); February 20-21, 2003. Claremont Hotel, Berkeley, CA, USA.

[5] J. Hill, R. Szewczyk, A.Woo, S. Hollar, D. E. Culler, and K. S. J. Pister. "System architecture directions for networked sensors", In Proc. the 9th International Conference on Architectural Support for Programming Languages and Operating Systems, Boston, MA, USA, Nov. 2000.

[6] S. Guha and E. G. Sirer, "Distributed Constraint-based Location Discovery in Ad hoc Networks," Cornell University, Tech. Rep. cul.cis/TR2004-1939, 2004.

[7] W. R. Heinzelman, A. Chandrakasan, and H. Balakrishnan, "Energy-Efficient Communication Protocol for Wireless Microsensor Networks," in Proceedings of HICSS, Jan. 2000.

[8] H. Z. Tan and I. Körpeoglu, "Power efficient data gathering and aggregation in wireless sensor networks," ACM SIGMOD Record, vol. 32, no. 4, pp. 66–71, Dec. 2003.

[9] R. Narayan, S. Guha and E. G. Sirer, "Position Informed Energy Efficient Sensing,". Under submission to SECON. 2004.

[10] Brad Karp and H. T. Kung, "GPSR: Greedy Perimeter Stateless Routing for Wireless Networks," in proceedings of International Conference on Mobile Computing and Networking, Aug. 2000.

[11] Janos Sallai, Gyorgy Balogh, Miklos Maroti and Akos Ledeczi, “Acoustic Ranging in Resource Constrained Sensor Networks,” Vanderbilt Unversity, Tech. Rep. ISIS-04-504, 2004.

[12] Aram Galstyan, Bhaskar Krishnamachari, Kristina Lerman and Sundeep Pattem, “Distributed Online Localization in Sensor Networks Using a Moving Target,” in proceedings of International Symposium on Information Processing in Sensor Networks, Apr. 2004.

[13] Jeremy Elson and Deborah Estrin, “An Address-Free Architecture for Dynamic Sensor Networks,” Computer Science Department USC, Tech. Rep. 00-724, 2000.