Ad-Hoc Smart Environments

Ramesh Govindan *Ted Faber* *John Heidemann* *Deborah Estrin*
USC/Information Sciences Institute
`{govindan,faber,johnh,estrin}@isi.edu`
`USC Technical Report 99-692`

1 Introduction

By early next century, sensor integration, coupled with unceasing electronic miniaturization, will make it possible to produce extremely inexpensive sensing devices. These devices will be able to monitor a wide variety of ambient conditions: temperature, pressure, humidity, soil makeup, vehicular movement, noise levels, lighting conditions, the presence or absence of certain kinds of objects, mechanical stress levels on attached objects, and so on. We expect that such devices will be equipped with significant processing, memory, and wireless communication capabilities.

Emerging low-level and low-power wireless communication protocols will enable us to *network* these sensors. This capability will add a new dimension to the capabilities of sensors: Sensors will be able coordinate amongst themselves on a higher-level sensing task, aggregating and averaging data to provide a more accurate estimate of their target (*e.g.,* reporting the exact speed, direction, size, and other characteristics of an approaching vehicle). Moreover, it improves overall accuracy and reduced costs through the use of many inexpensive, short-range sensors rather than a few expensive and complex long-range sensors.

Networking inexpensive sensors, and thereby creating *smart environments* can revolutionize information gathering in a variety of situations. In particular, one kind of smart environment that pushes the envelope of many systems design areas is what we call an *ad-hoc smart environment.* As an example of an ad-hoc smart environment, consider thousands of disposable sensors are densely scattered into an earthquake-damaged building. The sensors coordinate to establish a communication network, divide the task of mapping the damaged structure amongst themselves in an energy-efficient manner, adapt their overall sensing accuracy to the remaining total resources, and re-organize upon sensor failure. When additional sensors are added or old sensors fail, the sensors re-organize themselves to take advantage of the added system resources. Such a network can automatically, and rapidly, direct the nearest emergency response teams to wounded individuals, or find safe evacuation paths.

Ad-hoc smart environments impose two key challenges for communications and networking design: support for very large numbers of unattended autonomous nodes and adaptivity to environment and task dynamics.

Many large-scale networks exist today; the Internet is a prime example. Ad-hoc smart environments present a fundamentally more difficult problem, though, because the ratio of communicating nodes to users is much greater. Each personal computer on the Internet has a user who can resolve or at least report all manner of minor errors and problems. This human element allows the Internet to function with relatively brittle software. Ad-hoc smart environments, by comparison, will exist with the ratio of thousands of nodes per human user (or more). At such ratios, it is impossible to pay special attention to any individual node. Furthermore, even if it were possible to consider each node, sensors may be inaccessible, either because they are embedded in physical structures or thrown into inhospitable terrain. Thus, for such a system to be effective, it must provide *exception-free*, unattended operation (the term exception-free is due to Mark Weiser). Nodes must be completely self-configuring and robust to changes in condition.

It is not completely true that there are no large scale unattended systems today. Automated factories, for example, may contain hundreds largely unsupervised computers, but at the cost of careful pre-planning and design. This example illustrates the second requirement of ad-hoc smart environments: unlike factories, they must operate and respond to very dynamic environments without a carefully planned deployment. They will suffer substantial changes as nodes fail due to battery exhaustion or accident, new nodes are added, nodes move or are carried. User and environmental demands also contribute to dynamics as what is being sensed moves and what is considered interesting changes. Thus ad-hoc smart environments must automatically adapt to changes in environment and requirements.

One hypothesis for the overall design of such environments is that it is sufficient to design their applications using Internet technologies coupled with ad-hoc routing mechanisms. In such a design, each sensor node is an Internet-capable device (has one or more IP addresses) and can run applications and services. When deployed, sensor nodes establish an ad-hoc network amongst themselves; thereafter, application instances running on each node can communicate with each other. Applications, aided by directory and resource discovery services, are structured much the same way as traditional Internet applications.

We believe, however, that the requirements of ad-hoc smart environments are different enough from those of traditional wired and wireless networks to warrant considering a different design. This design has the following features:

Data-centric: Unlike traditional networks, a sensor node doesn't need an identity (*e.g.,* an address). That is, applications are unlikely to ask the question: What is the temperature at sensor #27? Rather, applications focus on the **data** generated by sensors. Data is named by attributes and applications request data matching certain attribute values. So, the communication primitive in this system is a request: Where are nodes whose temperatures recently exceeded 30 degrees? This approach decouples data from the sensor that produced it. This allows for more robust application design: even if sensor #27 dies, the data it generates can be cached in other (possibly neighboring) sensors for later retrieval.

Application specific: Traditional networks are designed to accommodate a wide variety of applications. We believe it is reasonable to assume that ad-hoc smart environments can be tailored to the sensing task at hand. In particular, this means that intermediate nodes can perform application-specific data aggregation and caching, or informed forwarding of requests for data. This is in contrast to routers that facilitate node-to-node packet switching in traditional networks.

Resource constrained: Traditional networks provide large bandwidths, wall power, and powerful compute elements. Sensor nodes will often be limited in one or all of these dimensions.

2 Research Issues

If we admit this hypothesis, several research questions arise, some of which we're actively exploring. We describe these in the following subsections.

2.1 Localized Algorithms

Centralized versions of sensor coordination applications have obvious robustness and scaling drawbacks. We believe, instead, that such applications are better realized using *localized* algorithms. We use this term to mean a distributed computation in which sensor nodes only communicate with sensors within some neighborhood, yet the overall computation achieves a desired global objective. What is the rationale for using localized algorithms in ad-hoc smart environments? Since the sensors themselves are physically distributed, it is not unnatural to design such environments using distributed algorithms. In many cases, the sensed activity itself is also localized. Finally, localized algorithms have two attractive properties. Because each node communicates only with other nodes in some neighborhood, the communication overhead *scales* well with increase in network size. For a similar reason these algorithms are *robust* to network partitions and node failures. We are just beginning the work of validating this hypothesis through design and experimentation.

Localized algorithms are hard to design for two main reasons. First, local algorithms must provide a desired global behavior with at best indirect global knowledge. Thus the process of crafting local algorithms from the global behavior is akin to the process of converting a centralized algorithm to a completely distributed one. The resulting rules often bear little resemblance to the original distributed computation. Second, some kinds of localized algorithms are parametrically sensitive; different choices of algorithm parameters can lead to radically different kinds of global behavior. An example of this is the reaction-diffusion systems studied by Turing. It is difficult to design localized algorithms that both empirically adapt to a wide range of environments and converge to the desired global behavior over that entire range.

2.2 Data-centric Communication

Another potential research topic is the design of abstractions that describe the communication patterns underlying most localized algorithms. To motivate such research, we briefly sketch one such set of abstractions, *directed diffusion.*

An ad-hoc smart environment based on directed diffusion exhibits the following properties. Each sensor node *names* data that it generates with one or more attributes. Other nodes may express *interests*, based on these attributes. Network nodes propagate interests. Interests establish *gradients* that direct the diffusion of data. As it propagates, data may be *cached or locally transformed* at each node. This aspect of the model leverages the application-specificity that is possible in smart environments. Caching and aggregation can increase the efficiency, robustness and scalability of coordination.

These abstractions capture the data-centricity and application-specificity inherent in smart environments. The model allows for neighbor-to-neighbor interest propagation and local data transformation rules; these elements capture the communication patterns expected of localized algorithms. Finally, gradients model the network-wide results of these local interactions. For these reasons, we believe this abstractions will allow us to express, and possibly reason about, a variety of localized algorithms.

2.3 Adaptive Fidelity Algorithms

One class of localized algorithms, that we call *adaptive fidelity* algorithms, are especially interesting because they are applicable to a wide variety of smart environment applications, and also have significant research potential. Such algorithms allow applications to vary the fidelity of sensing activity in task-specific ways. For example, an application may require only a few sensors to continuously monitor and detect background activity, and then *turn on* a larger collection of sensors upon perceiving the likelihood of a more significant threat.

The goal of this adaptation is to carefully utilize sensing resources in order to prolong network lifetime. These algorithms are essential in ad-hoc smart environments. In such environments, it will be impossible to determine, a priori, what size deployments are necessary. For this reason (and because we expect sensors themselves to be cheap), sensor network deployments will usually contain larger-than-necessary numbers of sensors. Without adaptive fidelity, these deployments can result in significantly reduced network lifetimes. For a similar reason, adaptive fidelity algorithms enable the network to work gracefully in the face of unplanned incremental deployment. Consider a collection of sensors "thrown" into an existing network. We cannot expect that the location of the "thrown" sensors will exactly augment the capacity of that part of the existing network that needs it: that is, many sensors may land near existing sensors that already have significant residual power. Adaptive capacity algorithms help use these sensors carefully to extend the overall lifetime of the network.

2.4 Designing for Localized Algorithms

Localized algorithms achieve a great degree of robustness through local interactions. However, these algorithms are not without their drawbacks. When deployed in a sensor network, however, localized algorithms may utilize more resources than necessary in order to achieve a given level of sensing fidelity. Intuitively, this may happen because nodes in a localized algorithm inherently "see" only local state; thus, they may be forced into local maxima or minima (resulting in reduced fidelity). To avoid this, the algorithm may attempt to perturb the system from these local maxima or minima, thereby expending more resources. Localized algorithms may also achieve lower sensing fidelity, given a particular resource utilization target. Roughly speaking, this may happen because these local interactions take more time to "diffuse" across the network before the desired global behavior is achieved. A third consequence of localized algorithms is their increased tendency towards pathological failures caused by rippling node failures or system-wide oscillations. Such behaviors are not unknown in highly distributed systems, like the Internet routing system.

To understand these tradeoffs, algorithm designers could greatly benefit from a design tool (analogous to a CAD-tool for VLSI-design) that, given any localized algorithm, can quantify its performance—i.e., its reduced fidelity, its increased responsiveness, or its increased susceptibility to pathological failures.

2.5 Operating Systems Issues

As with other fundamental changes in computing system organization, a shift to sensor networks requires changes to the operating system model. The basic entities that the OS manipulates must change from processes to data, which is reflected both in the interface presented to applications and in the internal mechanisms. Exception-free operation requires that the systems identify and neutralize faults in themselves and neighbors. Because the capabilities of sensor nodes will vary, usually directly with their importance, sensor-node operating systems must adapt to the underlying hardware.

Sensor nodes collect, manipulate, and forward constantly changing data, as compared to traditional computing systems which process files with comparatively static contents. Operating systems for sensor nets must reflect this fundamental shift. Resources allocation and accounting should be structured around classes of data objects rather than processes. Scheduling priority and caching will be based on data rather than applications. For example, acoustic data may be considered more accurate than seismic data, so the operating system may favor processing of acoustic data and cache a longer history of recent audio data. This shift of focus from processes to data simplifies the movement of information from remote sensors to interested computing engines, sharing of sensors allocated to multiple tasks, and supports collaboration between multiple sensors.

The focus on data must be reflected in the operating system interface. Traditional operating systems present a representation of sensing or display devices; a sensor-net operating system must represent data explicitly. An application requests that the operating system makes certain data available to it, rather than programming a sensor for output. By presenting a data-oriented interface, the application is insulated from the mechanisms used to gather that data. For example, data that is delivered to an application may be the direct output of one sensor or a synthesis of many sensor outputs; how the data was created can be transparent to the application.

Exception-free operation of the sensor net must be provided by the operating systems, because it is inherently cooperative and application-independent. A malfunctioning sensor node can affect not only its own output, but the functioning of the whole net. When a node fails, it may flood the network with incorrect data, disrupt data-forwarding systems, or disrupt routing protocols. Other more or less disruptive failure modes are possible.

A failing node may not be able to identify itself as malfunctioning, so its neighbors must participate in both identifying its failure and restricting any disruption it causes. This cannot be done by applications, because it requires access to the underlying system information, and collusion with sensor nodes that may not share an application.

The nodes in a sensor net can range from small sensors to capable workstation-class nodes. Small nodes must function effectively at single tasks, while larger nodes may be shared among several activities (perhaps concurrently summarizing and analyzing multiple data streams). All systems must provide a similar programming interface for data movement, but different levels of protection. Small, resource-limited nodes require minimal overhead and so will be dedicated to a single task and run without interprocess protection. Larger nodes will run multiple applications or parts of an application which should be protected from one another, both for security and robustness.

A sensor-net operating system must reflect its role as a data marshaller, and maintain the exception-free operation of the net. Accomplishing the first requires it to treat sensor data as a first-class system entity and export that view to applications. Operating the net reliably requires collusion among nodes to control failures and adapting to hardware resources to reduce failure probabilities.