IrisNet: An Architecture for a Worldwide Sensor Web

Today's common computing hardware—Internet connected desktop PCs and inexpensive, commodity off-the-shelf sensors such as Webcams—is an ideal platform for a worldwide sensor web. IrisNet provides a software infrastructure for this platform that lets users query globally distributed collections of high-bit-rate sensors powerfully and efficiently.

Phillip B. Gibbons and Brad Karp
Intel Research Pittsburgh

Yan Ke, Suman Nath, and Srinivasan Seshan
Carnegie Mellon University

Wide-area architectures for pervasive sensing will enable a new generation of powerful distributed sensing services. Such architectures have so far received little attention but are increasingly relevant because of a confluence of technological trends. Commodity off-the-shelf sensors—including video cameras (Webcams), microphones, and motion detectors—have become readily available. Such sensors interface easily with today's low-cost PCs, which are deployed globally and connect through the Internet. Indeed, we could even view a PC's high-speed network interface as a rich sensor that senses the virtual environment of a LAN or the Internet rather than the physical environment. Together, these devices present a promising hardware platform for an emerging wide-area sensor network. What's missing are the architecture, algorithms, and software system needed to orchestrate this hardware into a global sensor system that responds to users' queries.

We envision a *worldwide sensor web*, in which users can query, as a single unit, vast quantities of data from thousands or even millions of widely distributed, heterogeneous sensors. Internet-connected PCs that source sensor feeds and cooperate to answer users' queries will form the global sensor web's backbone. Developers of wide-area sensing services (*service authors*) will deploy their services on this distributed infrastructure.

Imagine the following scenario: after an oil spill, an ecologist wishes to know all locations where oil has significantly encroached on the coastal habitat. She queries a coastal-monitoring service, which collects data from video cameras directed at the coastline. In response, she receives both images of these contaminated sites and their geographic locations. The same coastal-monitoring service can store *triggered* queries, whereby the service notifies the appropriate lifeguards when a strong riptide develops in a beach region.

In the IrisNet (*I*nternet-scale *R*esource-*I*ntensive *S*ensor *Net*work Services) project at Intel Research, we are designing an architecture and building a system that enable easy deployment of such wide-area sensing services. Our aim is to provide the missing software components for realizing a worldwide sensor web.

Worldwide sensor web: context and benefits

To date, sensor-network research has largely been defined by the design of algorithms and systems to cope with the severe resource constraints of tiny battery-powered sensors that use wireless communication (for example, slow CPUs, low-bit-rate radios, and scarce energy). Such sensor networks are distributed over a single, contiguous communication domain. They use simple sensors that provide time series of single numerical measurements, such as temperature, pressure, light level, and so on. Researchers have developed spe-

Published by the IEEE CS and IEEE ComSoc ■ 1536-1268/03/$17.00 © 2003 IEEE

cialized hardware, operating systems, programming languages, and database systems to accommodate such severely constrained environments.[1–3]

In contrast, in IrisNet we seek to broaden the traditional notion of sensor networks to include wide-area "sensor webs,"[4] such as those comprising Internet-connected, widely-dispersed PC-class nodes with powerful CPUs that can process rich sensor data sources. A worldwide sensor web seamlessly integrates a wide range of sensor feeds, from high-bit-rate feeds from Webcam-equipped PCs to low-bit-rate feeds from traditional wireless sensor networks.

Additionally, a worldwide sensor web enables a wide variety of useful services. Consumer-oriented services include

- Alert services for notifying users when to head to the bus stop or when water conditions have become dangerous
- Waiting-time monitors for reporting on queueing delays at post offices, food courts, and so on
- Parking-space-finder services for directing drivers to available parking spaces near their destinations
- Lost-and-found services for locating lost objects or pets
- Watch-my-child (or watch-my-parent) services for monitoring children playing in the neighborhood (or elderly parents about town)

Other promising services exist in various domains. Epidemic early warning services (public health), homeland defense services (security), computer network monitoring services (technology), and Internet-scale sensing observatories (natural sciences) are a few possibilities.

Envisioning a worldwide sensor web

Wide-area sensing services have several key demands that drive worldwide sensor web design.

Planet-wide local data collection and storage

First and foremost, wide-area sensing necessarily employs numerous globally distributed sensing devices that observe the physical world. Because the sensors would collect a vast volume of data, and because we would want to retain both the most recent observations and a historical record, the system should store observations near their sources and transmit them across the Internet only as needed.

Real-time adaptation of collection and processing

The system should be able to reconfigure data-collection and -filtering processes in reaction to sensed data. Sampling rates might change; we might invoke new, special-purpose processing routines; we might even control actuators to modify data collection. Deciding when and how to adapt collection and processing might depend on sophisticated analysis of data derived from multiple sensors.

Data as a single queriable unit

The user should view the sensing device network as a single unit that supports a high-level query language. Each query would be able to operate over data collected from across the global sensor network, just as a single Google search query encompasses millions of Web pages. Beyond the keyword searches Google offers, the worldwide sensor web should support rich queries, which could include arithmetic, aggregation, and other database operators. (Google is not intended for sensing applications and so offers few of these features.)

Queries posed anywhere on the Internet

We are accustomed to retrieving information stored anywhere on the Internet from anywhere on the Internet. The sensor web should preserve this ubiquitous information access. At the same time, the system should actively seek to exploit any locality between the querier and the queried data. Because the sensed data are inherently coupled to a physical location, many queries will be posed within tens of miles of the data. For example, lifeguards responsible for monitoring a beach region would tend to request riptide alerts for the waters adjacent to "their" beach.

Data integrity and privacy

Pervasive monitoring of the physical world raises significant data integrity and privacy concerns. In our initial IrisNet prototype, we assume that the entire worldwide sensor web is administered by a single, universally trusted authority. In any real-world deployment of a global sensor web, different authorities will control portions of the sensing infrastructure, and sensor service authors might wish to compose services across authority boundaries.

Moreover, different service users might trust different authorities, as could different service authors. The sensor web should support defining and enforcing data integrity and privacy policies that match the concerns of four constituencies: service users, service authors, those in a sensed region, and those operating the sensor web. The combination of these policies should determine how data are distributed, processed, and shared in a sensor web.

Robustness

In a system that uses so many sensing devices and so many computing devices, failures will occur often. The system should operate smoothly despite these failures and the resulting unreachable hosts, unavailability of previously stored observations, missed new observations, and so on.

Ease of service authorship

Finally, the system should make it as easy as possible for service authors to develop sensing services by providing high-level abstractions of the sensing infrastructure that hide the complexities of the underlying distributed-data-collection and query-processing mechanisms. Ease of service authorship is crucial to fostering system adoption and proliferation of new services that use it.

Challenges for service authors

A service author deploying a wide-area sensing service faces several challenges to realizing the vision we just described. IrisNet can help ease service authorship.

Wide-area sensing's two most fundamental challenges are data collection and query answering. IrisNet provides software tools that automate these two processes in a service-neutral fashion so that several different services can be deployed on a single IrisNet software infrastructure.

Data collection

Although different services may seek different measurements from sensors, a subset of data collection tasks is common to all services. First, all services must instrument the environment with sensors. In a naive sensor deployment, each service author would need to deploy a separate sensor to monitor the same physical area. Requiring that separate sensors be deployed for each service, however would greatly increase the cost and configuration effort involved in deploying a service, hindering new service deployment. IrisNet shares sensors among multiple sensing services to maximize each deployed sensor's use to service authors. A particular sensor might be positioned to measure a physical area of interest for one particular service, but it might, in fact, provide data useful in multiple services.

While shared sensors simplify service deployment, they complicate resource management in the infrastructure. How do sensor owners allocate resources among services competing for the same sensor? How do they protect services running on the same sensor from one another? How do they avoid wasting computation if two sensing services overlap in the processing they perform on their input? The IrisNet architecture provides a runtime environment for sensor nodes that assists with these tasks. Moreover, sharing the sensor infrastructure across all services forces each service to select a set of sensor feeds relevant to that service. In the current IrisNet prototype, service authors learn about available deployed sensor feeds out-of-band of IrisNet. In the future, IrisNet will include a sensor feed discovery service to provide this system functionality.

A particular sensor might be positioned to measure a physical area of interest for one particular service, but it might, in fact, provide data useful in multiple services.

To support several rich sensor feeds, a service must reduce raw observations to postprocessed, extracted information near the data source. This filtering strategy avoids overloading the network and spreads the computational burden among many machines operating independently in parallel. The largest data reduction arises from using service-specific filtering. For example, filtering code for a coastal imaging service can reduce a 1.5-Mbps compressed video feed of a coastline to a single time-exposure image of 20–150 Kbytes every 10 minutes (under 2 Kbps). The IrisNet architecture makes it easy for services to upload such filtering code to sensing devices.

Querying the collected data

Sensing services typically collect filtered sensor readings in a database that users can query. For example, a coastal imaging service might use a database of all past single-frame images for monitored coastline locations, while a person-locator service might use a database of individual identifying information (for example, name or social security number) and location. Databases for different services might have grossly different properties. The database for an imaging service for an entire coast could be quite large, while that for a citywide person locator might be much smaller. Similarly, updates to sensed data for services such as a network and host monitor service that records every packet traversing a network's monitored portions might occur much more frequently than updates for services such as a parking-space finder and coastal imaging. Furthermore, the query patterns for distinct services might differ substantially. One would expect most parking-space-finder queries to be for neighborhoods near the querier, while queries for a coastal imaging service might cover large regions far from the querier. IrisNet provides tools that let service authors implement sensor-service databases. The key challenge in designing such tools is ensuring that they meet the wide range of real demands different services make.

Sensing services have a few common key properties that IrisNet's database support specifically addresses. First, although update rates for sensor databases can vary from service to service, they are often much greater than those in traditional databases. This occurs because data sources in a sensor data-

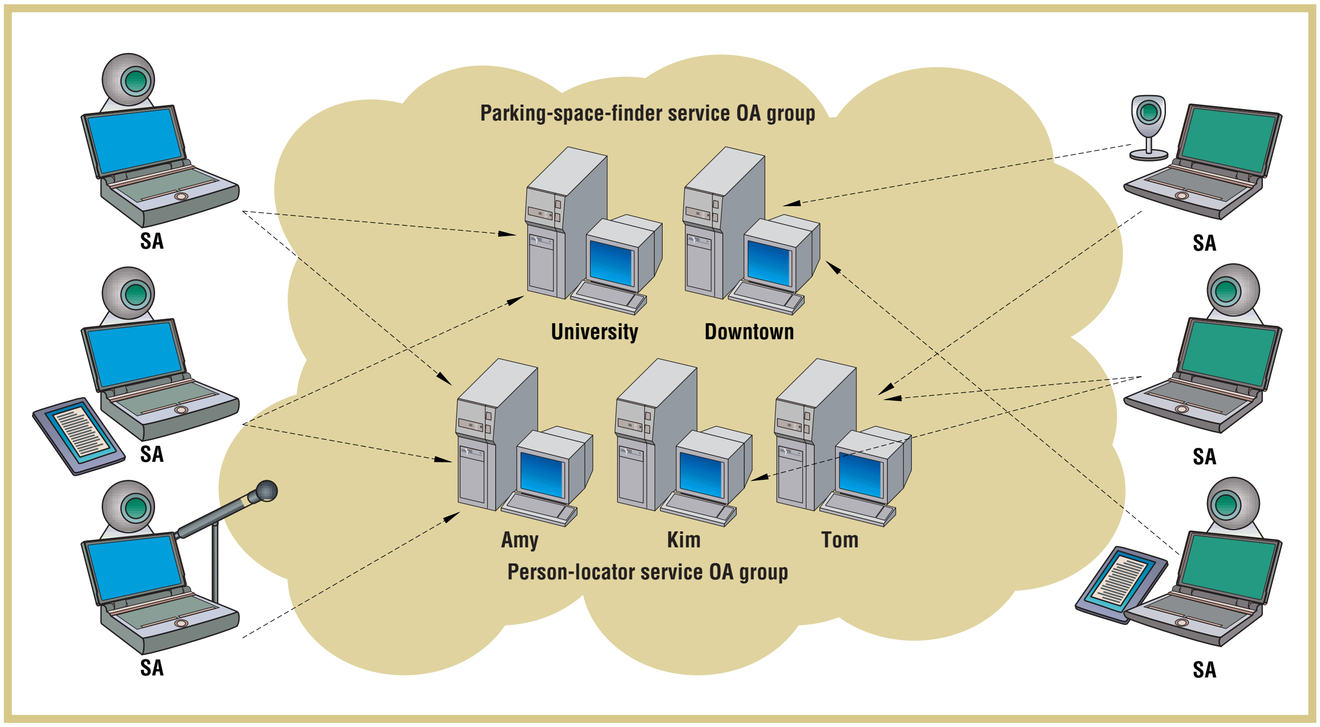

Parking-space-finder service OA group
University
Downtown
Amy
Kim
Tom
Person-locator service OA group
SA
SA
SA
SA
SA
SA

Figure 1. The IrisNet architecture, with sensing agent (SA) and organizing agent (OA) nodes.

base are automated, allowing for frequent updates without human intervention. Consider a person-locator service, which updates stored user positions once every 10 minutes. For a moderately sized metropolitan area of 1.5 million people, the user location database would need to process approximately 25,000 updates per second. The entire US (approximately 300 million people) would have approximately 500,000 updates per second. To support high update rates, we can partition the database across multiple host nodes. However, partitioning a database often limits query types or query performance. Fortunately, while all sensing services let users query the collected sensor readings, services also commonly target particular query types. For example, a coastal-imaging service might require queries to include a target location and date. Similarly, a person-locator service might need queries to specify a person's name and social security number. IrisNet gives service authors a hierarchically organized distributed database system tailored to these common characteristics. In addition, IrisNet automatically partitions the database among the hosts that participate in it and routes queries among these hosts.

The support IrisNet gives data collection and query processing drastically reduces the difficulty of authoring a wide-area sensing service. A service author using IrisNet need only write the code that filters sensor readings and the schema that define the database's contents. IrisNet's architecture implements this high-level abstraction of a sensing service for use by service authors.

The IrisNet architecture

Figure 1 shows IrisNet's two-tier architecture. The following observations motivated this design:

- Despite differences between sensor types, developers need a generic data acquisition interface to access sensors. In IrisNet, the nodes that implement this interface are called *sensing agents* (SAs).
- Services must store the service-specific data the SAs produce in a distributed database. In IrisNet, the nodes that implement this distributed database are called *organizing agents* (OAs).

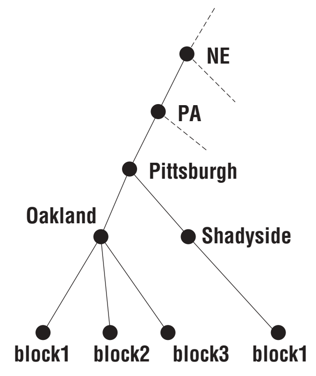
OA architecture

Service developers deploy sensing services by orchestrating a group of dedicated OAs. Consequently, each OA participates in only one sensing service (a single physical machine can run multiple OAs). The group of OAs for a single service must collect and organize sensor data to answer the particular class of queries relevant to the service (for example, queries about parking spots for a parking-space-finder service). OAs also should provide fault tolerance and balance load across the system. We use the parking-space-finder service to illustrate the important properties of IrisNet's query processing.

Choice of database. We envision a rich and evolving set of data types, aggregate fields, and so on within and across services, best captured by self-describing tags. In addition, each sensor reads a geographic loca-

```
<USRegion id="NE">
  <state id="PA">
    <city id="Pittsburgh">
      <neighborhood id="Oakland">
        <block id="block1">
          <parkingSpace>
            <handicapped>yes</handicapped>
            <available>yes</available>
          </parkingSpace>
        </block>
        <block id="block2">
          <parkingSpace>
            <available>yes</available>
          </parkingSpace>
        </block>
        <block id="block3">
          <parkingSpace>
            <available>yes</available>
          </parkingSpace>
        </block>
      </neighborhood>
      <neighborhood id="Shadyside">
        <block id="block1">
          <parkingSpace>
            <available>no</available>
          </parkingSpace>
        </block>
      </neighborhood>
    </city>
  </state>
</USRegion>
```

(a)



(b)

Figure 2. (a) Part of the XML schema used in the parking-space-finder service; (b) the hierarchy defined by the ID-tagged nodes.

```
/USRegion[@id='NE']/state[@id='PA']
  /city[@id='Pittsburgh']/neighborhood[@id='Oakland']
  /block[@id='block1' OR @id='block3']
  /parkingSpace[available='yes']
```

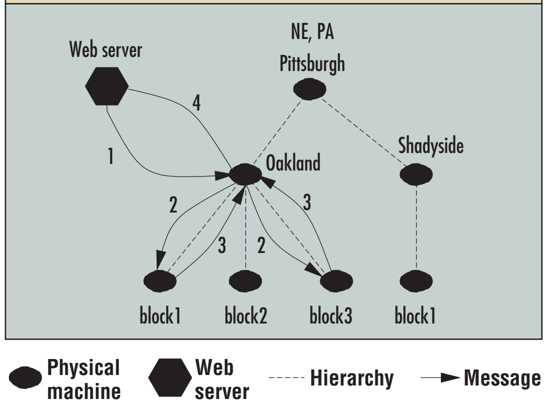


Figure 3. An example database configuration including (a) an XPATH query and (b) a mapping of logical nodes to seven machines and the messages sent between the machines to answer the query (numbers depict the relative order of messages).

tion, so it's natural to organize sensor data into a geographic- or political-boundary hierarchy. So, we choose to represent sensor-derived data in XML, which is well-suited to representing hierarchical data and uses self-describing tags to organize data. Figure 2a shows part of an XML document representing the parking-space-finder service's schema. The schema describes the static (for example, <handicapped>) and dynamic (for example, <available>) data as well as the hierarchy the service uses. The database schema the service author provides identifies the database's hierarchical portion using special ID tags. Figure 2b shows the hierarchy's tree representation, formed by the ID-tagged nodes.

Distributing the database. To adapt to query and update workloads, IrisNet dynamically partitions the sensor database among a collection of OAs. IrisNet permits an OA to own any subset of the nodes in the hierarchy (including noncontiguous subsets). It employs a distributed algorithm that—using several statistics that the OAs maintain—dynamically decides which parts of the sensor database should be partitioned or replicated and which OA to choose when placing database fragments. This adaptive data placement algorithm reduces average query response time and network traffic (for read and write operations) while keeping each OA's load below a load-limit threshold.

The path from the hierarchy's root to a lower node defines that lower node's globally unique name. (Achieving this property requires that the sibling nodes' ID tags be unique.) For example, the Pittsburgh node in Figure 2b is named city-Pittsburgh.state-PA.usRegion-NE. Each OA registers with the Internet's Domain Name System (DNS)[5] each node that it owns whose parent is owned by a different OA. The registered name is the node's global name prepended to the service's name and a registered domain suffix (for example, intel-iris.net). This is the only mapping from the logical database hierarchy to physical hosts' IP addresses in the system. This mapping allows considerable flexibility in mapping nodes in the XML document to OAs and OAs to physical machines. For example, Figure 3 shows an example configuration where one OA owns the nodes NE, PA, and Pittsburgh, and different OAs own each of the remaining nodes.

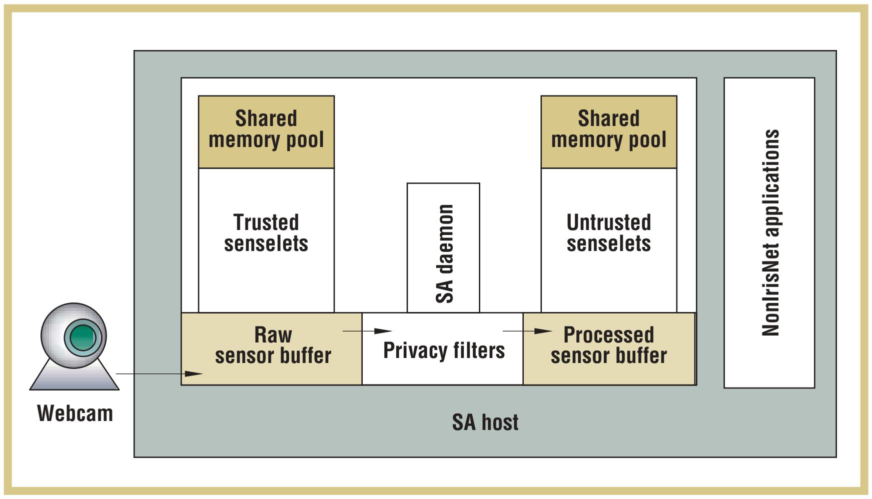


Figure 4. The execution environment in a sensing-agent host.

Query routing. We use the XPATH query language because it is the most widely used for XML, with good query processing support. Figure 3 shows an example XPATH query asking for all available parking spaces at block1 and block3 of Oakland.

Because our XML database is distributed, providing fast and correct answers to user queries is challenging. The system must route a query directly to the relevant nodes and must pass data between OAs only as needed. An XPATH query selects data from a node set in the hierarchy. In IrisNet, the query is routed directly to the *lowest common ancestor* (LCA) of the nodes the query potentially selects. For example, Oakland is the LCA node for the XPATH query Figure 3 shows. Note that we can derive the LCA node's globally unique name simply by parsing the XPATH query. A DNS lookup on this name provides the IP addresses of all OAs owning the LCA node. (As we discuss later, a node can be replicated, and thus owned by multiple OAs.) IrisNet selects one of these OAs, referred to as the LCA OA, and routes the query to that OA. This mechanism prevents the hierarchy's root from becoming a bottleneck; the LCA is typically far down in the hierarchy.

On receiving a query, the LCA OA queries its portion of the overall XML document and evaluates the result. For many queries, a single OA might not have enough of the document to respond. The OA determines which part of a user's query it can answer from the local document and where to gather the missing parts (extracting the needed global names from the document). The OA looks up other OAs' IP addresses and sends subqueries to them. These OAs might, in turn, perform a similar gathering task. Finally, the LCA OA collects the different responses and sends the combined result back to the user. For the example in Figure 3, the Oakland OA receives the query from the web server and sends subqueries to the block1 and block3 OAs. These each return a list of available parking spaces to be combined at the Oakland OA and returned to the user.

Caching and data consistency. To improve the performance of repeated requests for similar information (for example, multiple users requesting parking spaces downtown), OAs cache data from any query-gathering task that they perform. IrisNet's query-processing mechanism uses cached data even if the new query only partially matches the cached data. Using this cached information proves challenging because it complicates the OA's task of identifying the remaining information it must gather to answer a query.

Because of network propagation delays and the use of cached data, answers returned to users might not reflect the most recent database updates. In IrisNet, a query might specify consistency criteria indicating its tolerance for stale data. IrisNet stores timestamps along with the cached data that indicate when the data were created, so that an XPATH query specifying a tolerance automatically goes to data of appropriate freshness.

More details on IrisNet's database distribution, query processing, and caching techniques are available elsewhere.[6]

Fault tolerance. IrisNet replicates nodes in the logical hierarchy on multiple OAs. It uses two types of replicas: *primary replicas* that are kept strongly mutually consistent and placed in geographically optimal locations (for example, near the sensor reading sources) and *secondary replicas* that only maintain a weakly consistent copy of the data and are placed far from the primary replicas to maintain robustness when the primary replicas suffer correlated failure. During query routing, an OA generating a subquery to a target OA retrieves the list of primary replicas for the target OA using a DNS lookup and tries replicas sequentially until the query successfully reaches a live target OA. If the OA fails to find a live target OA, it performs a second DNS lookup to retrieve the list of secondary replicas, and again tries replicas sequentially. IrisNet uses a weighted probabilistic scheme that favors nearby replicas to select which replica to query next.

SA architecture

SAs collect raw sensor data from several sensors (possibly of different types). The sensor types can range from Webcams and microphones to temperature and pressure gauges. We focus our design on sensors such as Webcams that produce large volumes of data and that various services can use. Figure 4 shows the exe-

Related Work

Researchers have made various related efforts toward enabling wide-area sensing services. We can classify these into efforts with similar applications goals (sensor networks and video surveillance) and those that employ similar techniques (Internet service frameworks and distributed databases).

Sensor networks

Sensor networks and IrisNet share the goal of letting users access real-world measurements. The work on sensor networks has largely concentrated on using *motes*, small nodes containing a simple processor, a little memory, a wireless-network connection, and a sensing device. Due to the emphasis on resource-constrained motes, earlier key contributions have been in the areas of tiny operating systems[1] and low-power network protocols.[2] Mote-based systems have relied on techniques such as directed diffusion[3] to direct sensor readings to interested parties or long-running queries[4] to retrieve the needed sensor data to a front-end database. Other groups have explored using query techniques for streaming data and using sensor proxies to coordinate queries to address sensor motes' limitations.[5–7] None of this work considers sensor networks with intelligent sensor nodes, high-bit-rate sensor feeds, and global scale.

Video surveillance

Efforts such as the Video Surveillance and Monitoring project[8] have explored using video sensors. Such efforts have concentrated on image-processing challenges such as identifying and tracking moving objects in a camera's field of vision. These efforts are complementary to our focus on wide-area scaling and service authorship tools.

Internet services frameworks

Several efforts have produced frameworks for simplifying the development of scalable, robust Internet services.[9–12] In general, these projects target a lower level of the architecture than IrisNet. They concentrate on problems that are generic across all Internet services, such as load balancing, resource allocation, and network placement. In contrast, IrisNet addresses problems unique to services that must collect vast amounts of data and process queries on the data. In this way, IrisNet is largely complementary to these previous efforts and could, in fact, be implemented using these frameworks.

Distributed databases

IrisNet's distributed database infrastructure has much in common with various large-scale distributed databases. For example, the Domain Name System[13] relies on a distributed database that uses a hierarchy on the basis of host name structure to support name-to-address mapping. The Lightweight Directory Access Protocol[14] addresses some of the DNS's limitations by enabling richer standardized naming using hierarchically organized values and attributes. However, it still supports only a relatively restrictive querying model.

Matthew Harren and his colleagues have investigated peer-to-peer databases that provide a richer querying model than the exact-match queries existing distributed hash table (DHT) systems support. (They reported significant hotspots in storage, processing, and routing with their initial design.)[15] Replicated DHT-based databases and replicated hierarchical databases offer qualitatively different robustness behaviors. DHTs replicate portions of the DHT key space. Because DHT keys are typically hashes of data, failure of all replicas for the same portion of the DHT key space results in the disappearance of data items diffusely scattered throughout the database. In contrast, a hierarchical database replicates contiguous portions of its hierarchy across replicas. Should all replicas fail, all data beneath the corresponding contiguous portion of the hierarchy becomes unavailable. DHT-based databases are less well suited to leveraging XML and the hierarchically scoped queries typical in sensing services. Distributed databases supporting a full query-processing language such as SQL (Structured Query Language) are a well-studied topic,[16] with the focus on supporting distributed transactions or other consistency guarantees.[17–23] The previous work doesn't address the difficulties in distributed query processing over

cution environment in an IrisNet sensor host. A sensor host receives one or more raw sensor feeds from directly attached sensors and stores them in circular shared memory buffers. Multiple services can share an SA and access these buffers.

Programmable SAs. IrisNet lets services upload and control the execution of code that filters sensor readings dynamically in a service-specific fashion. We call this code a *senselet*. A single SA can execute one or more senselets for each service that wants to access its sensor feeds. A senselet instructs the SA to take the raw sensor feed, perform a specified set of processing steps (as the specific service requires), and send the resulting distilled information to a nearby OA. Although the OA receiving sensor readings from an SA will typically own the parts of the global sensor database that store the readings, this is not required. In the more general scenario, the nearby OA routes the update to the appropriate set of OAs owning the relevant portion of the database. Decoupling the SA that generates a sensor reading and the OA that owns the relevant database part provides a useful abstraction to those services that monitor mobile entities. For example, a person-locator service might use an organizational hierarchy to store people's locations (for an XML document. Researchers have also done considerable work on query processing over a federation of heterogeneous databases,[16] dealing with problems such as incompatible schemas. These problems do not arise in the current IrisNet architecture because the service author defines the schema for an entire service.

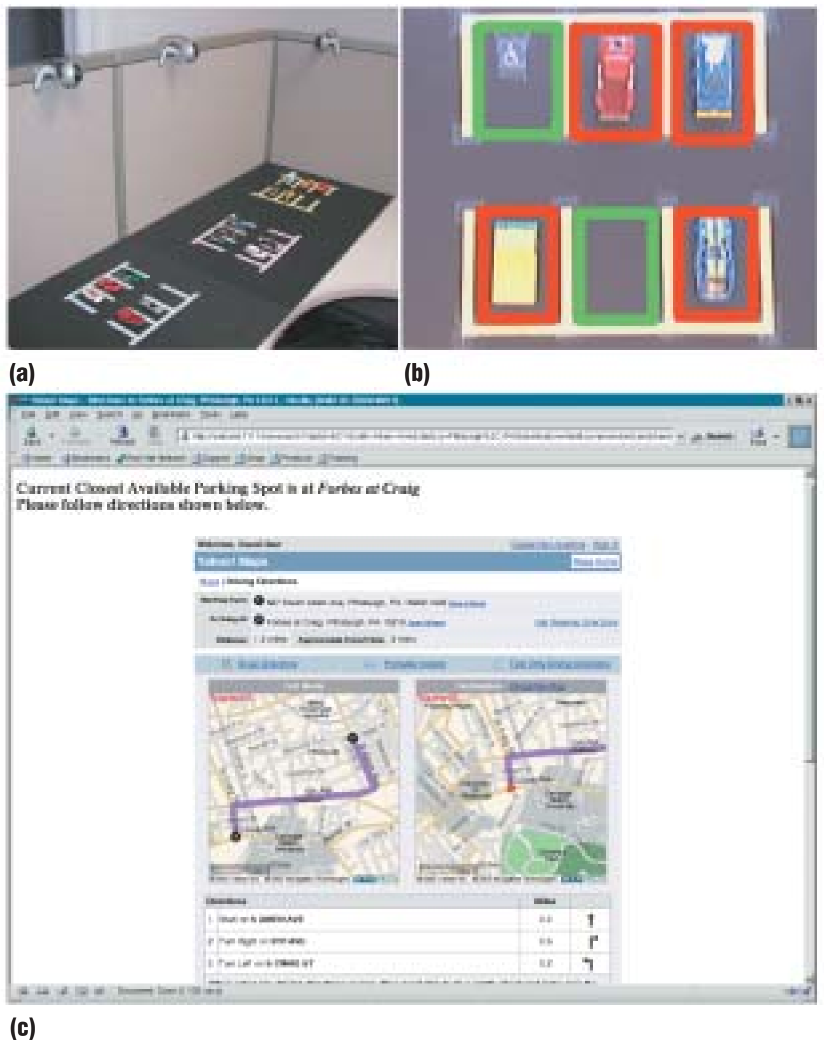
REFERENCES

1. J. Hill et al., "System Architecture Directions for Network Sensors," *Proc. 9th Int'l Conf. Architectural Support for Programming Languages and Operating Systems*, ACM Press, 2000, pp. 93–104.
2. J. Kulik, W. Rabiner, and H. Balakrishnan, "Adaptive Protocols for Information Dissemination in Wireless Sensor Networks," *Proc. 5th Ann. Int'l Conf. Mobile Computing and Networking*, ACM Press, 1999, pp. 174–185.
3. J. Heidemann et al., "Building Efficient Wireless Sensor Networks with Low-Level Naming," *Proc. 18th ACM Symp. Operating Systems Principles*, ACM Press, 2001, pp. 146–159.
4. P. Bonnet, J.E. Gehrke, and P. Seshadri, "Towards Sensor Database Systems," *Proc. 2nd Int'l Conf. Mobile Data Management*, LNCS, Springer-Verlag, 2001.
5. S. Madden and M.J. Franklin, "Fjording the Stream: An Architecture for Queries Over Streaming Sensor Data," *Proc. 18th Int'l Conf. Data Eng.*, IEEE CS Press, 2002, pp. 555–566.
6. S. Madden et al., "TAG: A Tiny Aggregation Service for Ad Hoc Sensor Networks," *ACM SIGOPS Operating Systems Rev.*, vol. 36, no. SI, Winter 2002, pp. 131–146.
7. S. Madden et al., "The Design of an Acquisitional Query Processor for Sensor Networks," *Proc. ACM Sigmod Int'l Conf. Management of Data*, ACM Press, 2003, pp. 491–502.
8. R. Collins et al., "Algorithms for Cooperative Multisensor Surveillance," *Proc. IEEE*, vol. 89, no. 10, Oct. 2001, pp. 1456–1477.
9. M. Welsh, D.E. Culler, and E.A. Brewer, "SEDA: An Architecture for Well-Conditioned, Scalable Internet Services," *Proc. 18th ACM Symp. Operating Systems Principles*, ACM Press, 2001, pp. 230–243.
10. J.R. von Behren et al., "Ninja: A Framework for Network Services," *Proc. Usenix Ann. Tech. Conf.*, Usenix, 2002.
11. *Libra: Scalable Advanced Network Services Based on Coordinated Active Components*, http://www-2.cs.cmu.edu/afs/cs.cmu.edu/project/cmcl/www/team.
12. I. Foster and C. Kesselman, *The Grid: Blueprint for a New Computing Infrastructure*, Morgan Kaufmann, 1999.
13. P.V. Mockapetris and K.J. Dunlap, "Development of the Domain Name System," *Proc. Symp. Communications Architectures and Protocols*, ACM Press, 1988, pp. 123–133.
14. M. Wahl, T. Howes, and S. Kille, *Lightweight Directory Access Protocol* (version 3), tech. report RFC 2251, Internet Engineering Task Force, 1997.
15. M. Harren et al., "Complex Queries in DHT-Based Peer-to-Peer Networks," *Proc. 1st Int'l workshop on Peer-to-Peer Systems*, LNCS, Springer-Verlag, 2002, pp. 242–250.
16. A. Silberschatz, H.F. Korth, and S. Sudarshan, *Database Systems Concepts*, McGraw Hill, 2002.
17. J. Sidell et al., "Data Replication in Mariposa," *Proc. 12th Int'l Conf. Data Eng.*, IEEE CS Press, 1996, pp. 485–494.
18. J. Gray et al., "The Dangers of Replication and a Solution," *Proc. ACM Sigmod Int'l Conf. Management of Data*, ACM Press, 1996, pp. 173–182.
19. C. Pu and A. Leff, "Replica Control in Distributed Systems: An Asynchronous Approach," *Proc. ACM Sigmod Int'l Conf. Management of Data*, ACM Press, 1991, pp. 377–386.
20. D. Agrawal and S. Sengupta, "Modular Synchronization in Distributed, Multiversion Databases: Version Control and Concurrency Control," *IEEE Trans. Knowledge and Data Eng.*, vol. 5, no. 1, 1993, pp. 126–137.
21. N. Krishnakumar and A. Bernstein, "Bounded Ignorance in Replicated Systems," *Proc. Symp. Principles of Database Systems*, ACM Press, 1991, pp. 63–74.
22. R. Alonso, D. Barbara, and H. Garcia-Molina, "Data Caching Issues in an Information Retrieval System," *ACM Trans. Database Systems*, vol. 15, no. 3, Sept. 1990, pp. 359–384.
23. S.W. Chen and C. Pu, *A Structural Classification of Integrated Replica Control Mechanisms*, tech. report CUCS-006-92, Columbia Univ. 1992.

example, people in academic departments in universities). When a participant in the service travels, he or she will pass within the view of sensors whose SAs might have no *a priori* association with the OAs that own the database node containing his or her location. Each such SA sends the updated location information to a nearby OA. IrisNet routes that update to the OAs owning the node.

Protecting senselets and SA hosts. The SA execution environment protects the host and senselets from buggy or malicious senselets. Each senselet executes as a different process. This separation provides process-level protection between senselets. It also enables limiting the resources that a senselet consumes, although our current implementation doesn't enforce such limits. This lack of resource limitations would not significantly hinder deployments in which service developers pay service providers for the resources consumed.

Privacy mechanisms. Video streams from cameras in public places will often contain human faces, automobile license plates, and other data that can be used to identify a person. This disclosure of identifying information raises an important question: how can IrisNet help protect



Current Closest Available Parking Spot is at *Forbes at Craig*
Please follow directions shown below.

Figure 5. (a) Mock parking lots, (b) postprocessed video stream of one parking lot, and (c) driving directions to the parking spot.

people's privacy? Ensuring with full generality that no one can use a video stream to compromise another individual's privacy is out of scope for our work on IrisNet. Nevertheless, we believe that IrisNet must provide a framework to help limit senselets' ability to misuse video streams for unauthorized surveillance. To this end, our current SA implementation uses privacy filters to identify the sensitive components (for example, human faces) in an image and replaces these regions with black rectangles. System administrators can load arbitrary privacy filters to enforce policies that are appropriate for their sites. As Figure 4 shows, IrisNet distinguishes between *trusted* and *untrusted* senselets; trusted senselets receive raw video feeds, whereas untrusted ones receive only the privacy-filtered video data. IrisNet uses digital signatures to authenticate trusted senselets to ensure that if sensitive information is leaked, system administrators can discover who is responsible.

Shared computation among senselets. The filtering that senselets do is computationally expensive. To increase the efficiency of executing multiple senselets on the same SA, we exploit the fact that one sensor feed might interest multiple, different IrisNet services. For example, a video feed in a particular location might monitor parking spaces in one service and track passersby in the same visual field in another. Senselets working on the same video stream often use many of the same image-processing primitives (for example, color-to-gray conversion, noise reduction, edge detection, maintaining a statistical background model, and so on). IrisNet provides a limited memoization mechanism that senselets can use to share intermediate computational results with other senselets through a shared memory pool. Intermediate results are named using the sequence of function calls performed on them, where each function is from a common library of image processing primitives. Each function call first checks whether the desired result already exists in the shared memory.

Prototype IrisNet applications

Along with our collaborators (in the case of the coastal imaging service) we have built three services from different application domains using IrisNet. Here we describe their implementation and deployment.

Parking-space finder

The parking-space finder uses cameras throughout a metropolitan area to track parking space availability. Users fill out a Web form to specify a destination and any constraints on a desired parking space (for example, does not require a permit, must be covered, and so on). Based on the input criteria, the parking-space-finder service identifies the nearest available parking space that satisfies the user constraints, then uses the Yahoo! Maps service to find driving directions to that parking space from the user's current location.

Although our current prototype operates on mock parking lots and miniature cars laid out on a table top, the rest of the system operates as it would in an outdoor setting. Senselets that recognize parking space availability run on laptops with Webcams. After the reference background image is initially calibrated, each senselet detects a car's presence by comparing the current image of the spaces with the corresponding background images.[7] The senselets do these image processing tasks using the Intel Open Source Computer Vision library (www.intel.com/research/mrl/research/opencv) provided with the SA environment. Once a senselet determines which parking spaces are empty, it sends the availability information to the appropriate OAs.

Figure 2 shows part of the database schema for this service. As mentioned earlier, this schema defines the database's geographically hierarchical organization and the dynamic (for example, availability information from the SAs) and static (for example, handicapped or not) descriptions for each space.

Figure 5 shows our laboratory deployment of this service on IrisNet. Figure 5a shows the mock parking lots, monitored by Webcams. Figure 5b shows the view of one parking lot after local image processing to recognize full and empty parking spaces; red rectangles indicate full spaces, while green rectangles indicate empty ones. Figure 5c shows an example of the driving directions the user receives. These directions are continually updated as the user drives toward the destination, if the parking spot availability changes, or if a closer parking spot satisfying the user's constraints becomes available.

Network and host monitor (IrisLog)

The IrisLog service collects data from host and network monitoring tools (which act as sensing devices) running on a widely dispersed set of hosts and lets users query those data efficiently. IrisLog is deployed on PlanetLab,[8] an open, shared, planetary-scale application testbed comprising hundreds of nodes distributed across five continents (see Figure 6). Using a Web-based form, users can query sets of PlanetLab nodes (for example, all the nodes at Carnegie Mellon University) by particular metrics (for example, CPU load) and time periods over which the data have been collected (for example, the last hour). The Web user interface also lets users issue arbitrary XPath queries over the distributed XML database of all IrisLog data.

IrisLog supports a superset of the queries from the Ganglia PlanetLab monitoring service,[9] but incurs far less network overhead. Each PlanetLab node runs an SA, which uses the local Ganglia daemon output to create a sensor feed describing 30 different performance metrics. The senselet for IrisLog transmits these metrics to the matching OA in the schema.

The IrisLog schema describes the

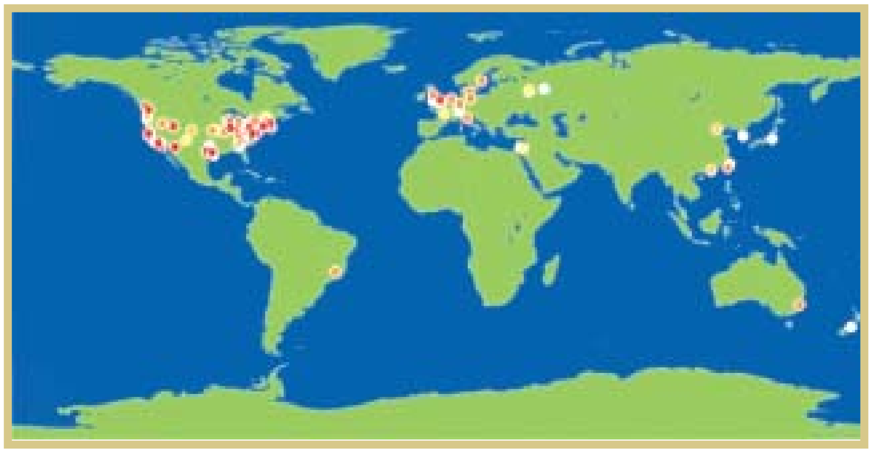
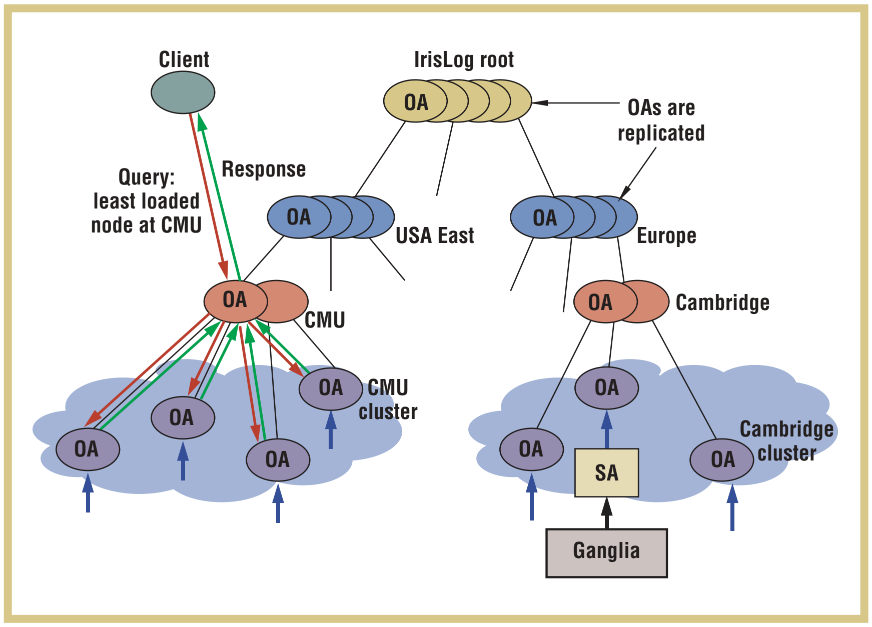


Figure 6. Deployed PlanetLab nodes as of October 2003.

Figure 7. A portion of the IrisLog hierarchy.



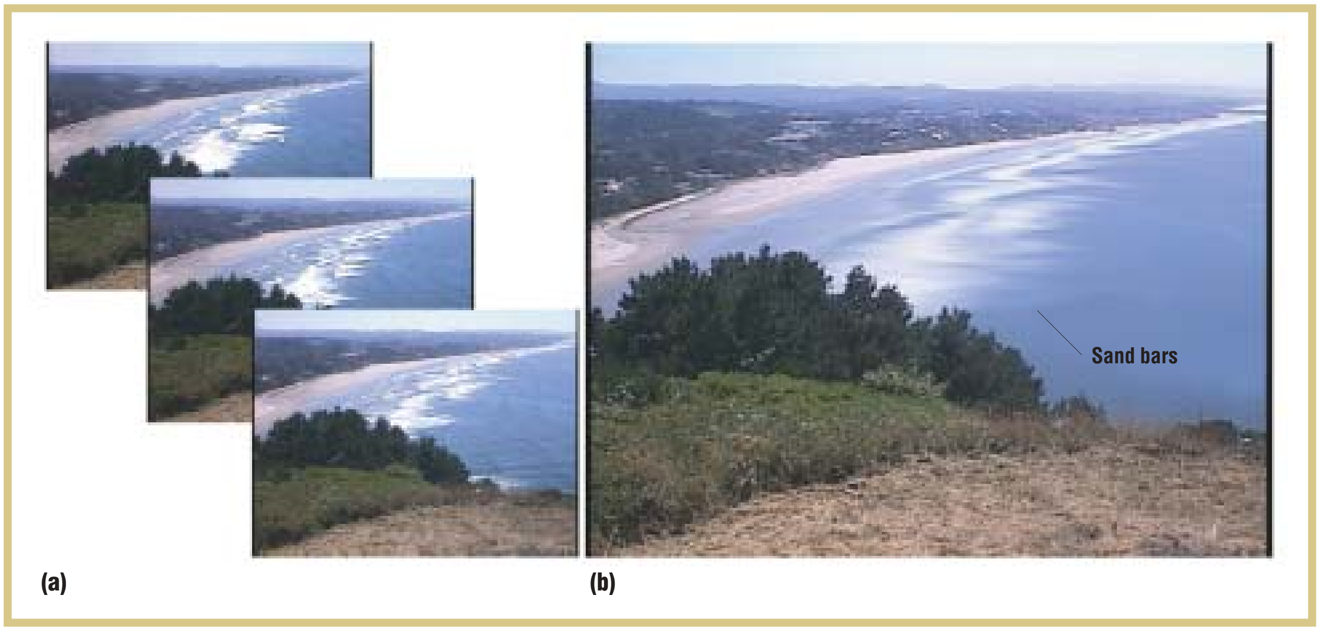


Figure 8. Coastal images in the form of (a) raw video frames and (b) 10-minute time-averaged exposure.

metrics each PlanetLab node must monitor (for example, CPU and memory load, bandwidth usage, and so on) and organizes them into a geographical hierarchy. This hierarchy (see Figure 7) allows efficient processing of geographically scoped queries (for example, find the least loaded CMU node). To support simple historical queries over the monitored data, the schema uses multiresolution vectors to store each monitored metric. These vectors provide samples of past data where the sampling rate is higher for recent data than for older data.

Coastal imaging service

In collaboration with oceanographers of the Argus project at Oregon State University, we have deployed a coastal imaging service on IrisNet (see the Applications department on page 14).[10] The service uses cameras installed at sites along the Oregon coastline and processes the live feed from the cameras to identify the visible signatures of nearshore phenomena such as riptides, sandbar formations, and so on. For example, Figure 8 shows how we can use the time-averaged exposures of camera images shown in figure 8b, generated by merging the raw frames in Figure 8a, to identify sandbar formations. The front-end of this service lets users query this historical information distributed across multiple sites. Users can change collection and filtering parameters remotely and install triggers to change the data acquisition rate after certain events (for example, darkness).

The senselet for this service produces image data such as 10-minute interval snapshots, 10-minute time-averaged exposure images that show wave dissipation patterns (indicating submerged sandbars), variance images, and photogrammetric metadata.

The schema defines a shallow hierarchy comprising a root node with each coastal camera location connected as leaf nodes. Each leaf node stores only the most recent snapshot and 10-minute time exposure, while the root stores an archive of all past snapshots from all the coastal cameras. These nodes are mapped onto physical OAs such that the root is located at a computer at Oregon State and the leaf OAs are located at the corresponding coastal locations.

Sensor network research has largely focused on localized deployments of low-power wireless sensors collecting numerical measurements. However, as our examples show, many sensing services require a wide-area network of powerful sensors such as video cameras. IrisNet is a general-purpose software infrastructure that supports the central tasks common to these services: collecting, filtering, and combining sensor feeds, and performing distributed queries within reasonable response times.

We are far from the vision of a highly available, high-performance, easy-to-use worldwide sensor web. While IrisNet represents an important first step toward this vision, its design focuses on the technical challenges we've described. Important policy, privacy, and security concerns must be addressed before rich sensors can exist pervasively at a global scale. Moreover, sensors must be deployed and maintained. Currently, companies such as Honeywell and ADT provide sensor-based home security systems. We envision that in time, such private-sector companies, in combination with public-sector entities, nonprofits, and individuals, might provide the needed sensor infrastructure.

ACKNOWLEDGMENTS

We thank Amol Deshpande, Rahul Sukthankar, and Shimin Chen for their important contributions to the IrisNet system. We thank Mahadev Satyanarayanan for many valuable suggestions and Casey Helfrich for helping with the experimental setup. Finally, we thank Mark Abbott, Rob Holman, Chuck Sears, Ganesh Gopalan, and Curt Vandetta for their collaboration on the coastal-imaging service.

REFERENCES

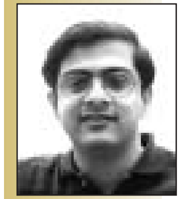
1. J. Hill et al., "System Architecture Directions for Network Sensors," *Proc. 9th Int'l Conf. Architectural Support for Programming Languages and Operating Systems*, ACM Press, 2000, pp. 93–104.
2. J. Kulik, W. Rabiner, and H. Balakrishnan, "Adaptive Protocols for Information Dissemination in Wireless Sensor Networks," *Proc. 5th Ann. Int'l Conf. Mobile Computing and Networking*, ACM Press, 1999, pp. 174–185.
3. S. Madden et al., "TAG: A Tiny Aggregation Service for Ad Hoc Sensor Networks," *ACM SIGOPS Operating Systems Rev.*, vol. 36, no. SI, Winter 2002, pp. 131–146.
4. Open GIS Consortium, *Sensor Web Enablement and OpenGIS SensorWeb*, www.opengis.org/functional/?page=swe.
5. P.V. Mockapetris and K.J. Dunlap, "Development of the Domain Name System," *Proc. Symp. Communications Architectures and Protocols*, ACM Press, 1988, pp. 123–133.
6. A. Deshpande et al., "Cache-and-Query for Wide Area Sensor Databases," *Proc. ACM SIGMOD Int'l Conf. Management of Data*, ACM Press, 2003, pp. 503–514.
7. L.G. Shapiro and G.C. Stockman, *Computer Vision*, Prentice Hall, 2001.
8. L. Peterson et al., "A Blueprint for Introducing Disruptive Technology into the Internet," *Proc. 1st Workshop Hot Topics in Networks* (Hotnets-I), ACM Press, 2002.
9. M.L. Massie, B.N. Chun, and D.E. Culler, "The Ganglia Distributed Monitoring System: Design, Implementation, and Experience," submitted for publication, Feb. 2003; http://ganglia.sourceforge.net.
10. R. Holman, J. Stanley, and T. Ozkan-Haller, "Applying Video Sensor Networks to Nearshore Environment Monitoring," *IEEE Pervasive Computing*, vol. 2, no. 4, Oct.–Dec. 2003, pp. 14–21.

the AUTHORS

Phillip B. Gibbons is a principal research scientist at Intel Research Pittsburgh and an adjunct professor in the computer science departments at Carnegie Mellon University and the University of Pittsburgh. His research interests include wide-area sensing, massive data sets, data streams, query processing and optimization, high-performance databases, approximate query answering, models of parallel computation, verification and testing, multiprocessor cache protocols, multiprocessor scheduling, and distributed computing. He received his PhD in computer science from the University of California at Berkeley. He is on the editorial board for the *Journal of the ACM*, and he is Conference Chair for the ACM Symposium on Parallelism in Algorithms and Architectures. He is a member of the ACM, the IEEE, and SIAM. Contact him at phillip.b.gibbons@intel.com.

Brad Karp is a staff researcher at Intel Research Pittsburgh and an adjunct assistant professor of computer science at Carnegie Mellon University. His research interests include algorithms and systems for routing, congestion control, and distributed storage in sensor networks, multihop wireless networks, and the Internet. He received his PhD in computer science from Harvard University. He is a member of the ACM. Contact him at brad.n.karp@intel.com.

Yan Ke is a graduate student at Carnegie Mellon University. His current research interests are computer vision and computer systems. He received his MS from Carnegie Mellon University. Contact him at yke@cmu.edu.


Suman Nath is a PhD candidate in the Computer Science Department at Carnegie Mellon University. His research interests include distributed systems, sensor networks, database systems, and networking. He received his MS in computer science from Carnegie Mellon University. He is a member of the ACM. Contact him at sknath@cs.cmu.edu.

Srinivasan Seshan is an associate professor in Carnegie Mellon University's Computer Science Department. His primary interests are in the broad areas of network protocols and distributed network applications. His current work explores new approaches in overlay networking, sensor networking, online multiplayer games and wide-area Internet routing. He received his PhD from the Computer Science Department at the University of California at Berkeley. Contact him at srini@cmu.edu; www.cs.cmu.edu/~srini.