IrisNet: An Architecture for Internet-Scale Sensing

Phillip B. Gibbons[*] Brad Karp[*] Yan Ke[†,*] Suman Nath[†,*] Srinivasan Seshan[†,*]

[*]*Intel Research Pittsburgh* [†]*Carnegie Mellon University*

1 Introduction

The time has come to consider *wide-area architectures* for pervasive sensing. Today's low-cost PCs are deployed globally, connected to the Internet, and routinely have diverse powerful sensors attached to them, such as video cameras (webcams) and microphones. Even a PC's high-speed network interface is a rich sensor, but one that senses the virtual environment of a LAN or the Internet, rather than the physical environment. Taken in the aggregate, the aforementioned hardware can be seen as an emerging global-scale sensor network. However, this sensor network lacks the architecture, algorithms, and software system needed to make it respond to users' queries.

We envision a global sensing system comprised of such rich, high-bit-rate sensor feeds—in which voluminous sensed data streams from vast collections of widely distributed sensors are available to users for querying as a single unit. To illustrate the type of wide-area sensing service we envision, imagine driving towards a destination in a busy metropolitan area. While stopped at a traffic light, you query a Parking Space Finder service using your PDA, by specifying your destination and criteria for desirable parking spaces (*e.g.,* within two blocks of your destination, at least a four-hour meter). You get back directions to an available parking space satisfying your criteria. A half hour later, you return to your car and discover that it has been dented! By querying an Accident Witness service using your PDA, you retrieve images showing how your car was dented and by whom.

These two example applications, a Parking Space Finder service and an Accident Witness service, support user queries over large collections of widely distributed video streams. In the IrisNet (Internet-scale Resource-Intensive Sensor Network Services) project at Intel Research Pittsburgh, we are designing and building an architecture and system that enable easy deployment of such wide-area sensing services over rich data feeds.

To date, sensor network research has largely been defined by the design of algorithms and systems to cope with the severe resource constraints of battery-powered, tiny sensors which use wireless communication—slow CPUs, low-bit-rate radios, and scarce energy. Such sensor networks are deployed over a single, contiguous communication domain. They use simple sensors that provide time series of single numerical measurements, such as temperature, pressure, light level, *etc.* Specialized hardware, operating systems, programming languages and database systems have been developed to accommodate this severely constrained environment [13, 16, 18].

In IrisNet, we seek to broaden the definition of "sensor network" to include the important, complementary class of wide-area sensor networks—Internet-connected, widely-dispersed, PC-class nodes with powerful CPUs that can process rich sensor data sources. Indeed, such a wide-area sensing architecture is composable with existing low-power sensor networks: IrisNet is equally adept at integrating video streams from webcam-equipped PCs as at integrating sensed data from distinct, widely dispersed clouds of low-power wireless sensors.

In this article, we offer a vision for wide-area sensing services; enumerate the technical challenges in achieving this vision; describe the IrisNet architecture, and how it meets these challenges; and report on present-day uses of the IrisNet system in several real wide-area sensing applications.

2 A Vision for Internet-Scale Sensing

As a broadly applicable architecture for Internet-scale sensing, IrisNet must meet many key demands of sensing applications:

- **Planet-wide local data collection and storage:** First and foremost, Internet-scale sensing entails multitudes of sensing devices spread across the planet collecting observations of the physical world. Because the total volume of data collected is vast, and because it is desirable to retain not only the most recent observations, but also a record of the past, observations must be stored near their source.

- **Real-time adaptation of collection and processing:** Data collection and filtering processes must be reconfigurable in reaction to the sensed data themselves—sampling rates may be changed; new, special-purpose processing routines may be invoked; actuators may even be controlled to modify data collection. Note that the decision of whether and how to adapt collection and processing may be based on sophisticated analysis of data derived from multiple sensors.

- **Data as a single queriable unit:** The network of sensing devices must appear to the user as a single unit, and must support a high-level query language. Each query may 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 offered by Google, the wide-area sensing architecture must support rich queries, which may include arithmetic, aggregation, and other database operators.[1]

- **Queries posed anywhere on the Internet:** We are accustomed to retrieving information stored anywhere on the Internet from anywhere on the Internet. Internet-scale sensing should preserve this ubiquitous information access. At the same time, the architecture should actively seek to exploit any locality between the querier and the data being queried. Because of the inherent coupling of sensed data to a physical location, most queries will be posed within tens of miles of the data. For example, requests for available parking spaces will be made by drivers within a few miles of the desired spaces.

- **Robustness:** In a system that makes use of so many sensing devices and so many computing devices, failures will occur often, and the system must operate smoothly despite these failures, missed observations, *etc.*

Internet-scale sensing supports a variety of useful sensor-enriched services. We have developed a prototype of the previously mentioned Parking Space Finder service (see sidebar). Other consumer-oriented services include alert services, such as a Bus Alert service, for notifying users when to head to the bus stop; waiting time monitors for reporting on queueing delays at post offices and food courts; lost and found services for locating lost objects or lost pets; and watch-my-child (or watch-my-parent) services for monitoring one's children playing in the neighborhood (or one's elderly parents about town). Other services in a wide variety of domains hold promise, *e.g.,* epidemic early warning services (public health), homeland defense services (security), computer network monitoring services—see sidebar (technology), and Internet-scale sensing observatories (natural sciences) are but a few possibilities. As an example of the latter, we are collaborating with a team of oceanographers at Oregon State University to develop a coastal imaging service on IrisNet (see sidebar).

3 Challenges for Service Authors

For a programmer who wants to deploy a wide-area, Internet-scale sensing service, several challenges arise on the way to realizing the vision described in the previous section. In this section, we describe how IrisNet eases the authoring of an Internet-scale sensing service.

There are compelling reasons why a sensor must be *shared* by multiple sensing services. A particular sensor may be located such that it measures a physical area of interest to a particular service author. But one such sensor may in fact provide data useful to *multiple* services. In a naive sensor deployment, each service author would be required to deploy a separate sensor to monitor the same physical area. The associated cost and configuration effort for separate sensors for each service would likely limit the extent of sensor deployment, and thus hinder service deployment.

To reduce cost and configuration effort, IrisNet shares sensors among multiple applications. This sharing poses several challenges to the service author. How does she allocate resources among competing services that use the same sensor? How does she protect services running on the same sensor from one another? How does she avoid wasting computation if two sensing services overlap in the processing they perform on their input?

Perhaps the two most fundamental challenges in wide-area sensing are the collection of widely distributed data and answering of queries over the collected data. IrisNet provides software tools that automate these two processes in a service-neutral fashion, so that a wide variety of services can be deployed on a single IrisNet software infrastructure.

First, let us consider the collection of data. While different services may seek different measurements from sensors, there remains a subset of data collection tasks common to all services. As a first step, each service must first select a set of sensor feeds that are relevant to that service. For example, a Person Locator service may desire a set of video sensor feeds from cameras that cover a campus or metropolitan area, whereas a Parking Space Finder service may only need feeds from cameras with views of parking lots.

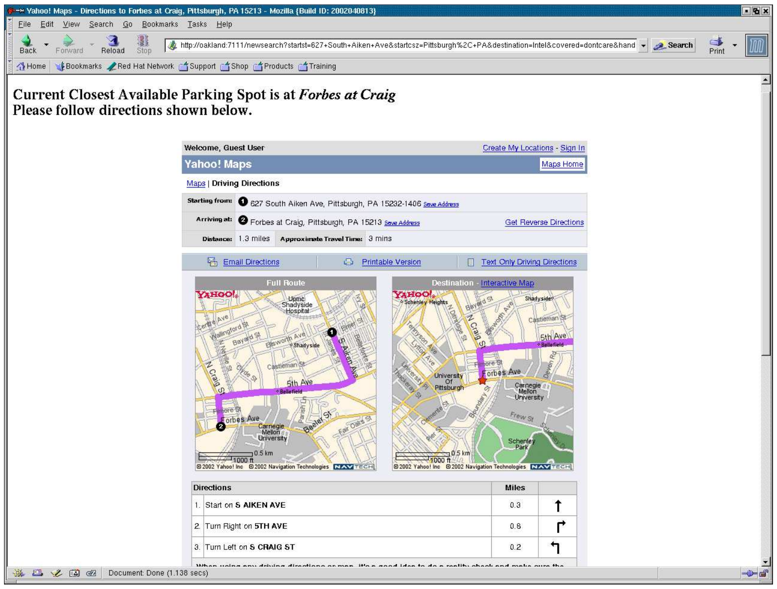
For scalability when dealing with rich data sources, raw observed data must be reduced to derived information immediately. Performing this processing near the source of the data avoids overloading the network, and spreads the computational burden among many machines operating independently in parallel. The largest data reduction arises from using service-specific filtering. For example, filtering code for a Parking Space Finder service can reduce a compressed 300 Kbps video feed of a parking lot to a few bits per second of parking space availability information, by processing the video feed to recognize empty and full parking spaces in the lot. The IrisNet architecture makes it easy for services to upload filtering code to sensing devices.

Next, let us consider querying the collected data. Sensing services typically collect filtered sensor readings into a database which users can query. For example, a Parking Space Finder service may use a database of parking spot properties and availability, while a person locator service

[1]Google is not intended for Internet-scale sensing, and hence offers few of these features.

Parking Space Finder

The Parking Space Finder uses cameras throughout a metropolitan area to keep track of the availability of parking spaces. Users fill out a Web form to specify a destination, and any constraints on a desired parking space (*e.g.*, does not require a permit, must be covered, *etc.*). Based on the input criteria, the Parking Space Finder service identifies the nearest available parking space that satisfies the user constraints, and then uses the Yahoo! Maps Service to find driving directions to that parking space from the user's current location. The figure below shows an example of the driving directions that are then displayed to the user. These driving directions are continually updated as the user drives towards the destination, if the availability of the parking spot changes, or if a closer parking spot satisfying the constraints is available. We envision that a car navigation system will periodically repeat the query as the user nears the destination.


Yahoo! Maps - Directions to Forbes at Craig, Pittsburgh, PA 15213 - Mozilla {Build ID: 2002040813}
File Edit View Search Go Bookmarks Tasks Help
Back Forward Reload Stop
Search Print
Home Bookmarks Red Hat Network Support Shop Products Training
Welcome, Guest User
Create My Locations - Sign In
Yahoo! Maps
Maps Home
Maps | Driving Directions
Email Directions
Printable Version
Text Only Driving Directions
Full Route
Destination - Interactive Map
Save Address
Get Reverse Directions
Document: Done (1.138 secs)

http://oakland:7111/newsearch?startst=627+South+Aiken+Ave&startcsz=Pittsburgh%2C+PA&destination=Intel&covered=dontcare&hand

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

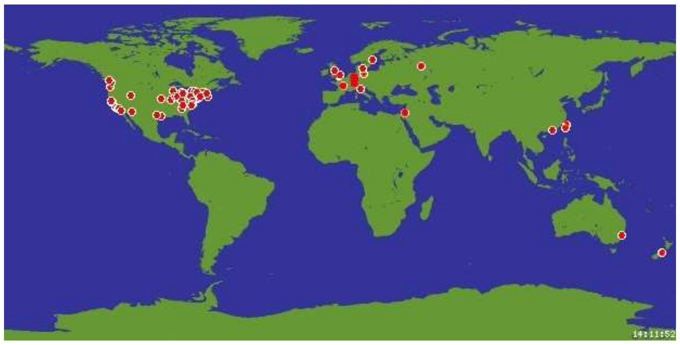
Starting from:	1 627 South Aiken Ave, Pittsburgh, PA 15232-1406
Arriving at:	2 Forbes at Craig, Pittsburgh, PA 15213
Distance:	1.3 miles
Approximate Travel Time:	3 mins

Directions		Miles	
1.	Start on **S AIKEN AVE**	0.3	↑
2.	Turn Right on **5TH AVE**	0.8	↱
3.	Turn Left on **S CRAIG ST**	0.2	↰

Driving directions to the parking spot are displayed.

Network and Host Monitoring

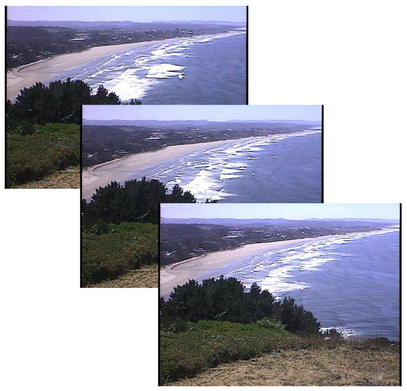
The IrisLog (`www.intel-iris.net/irislog.html`) service collects data from host and network monitoring tools (which act as sensing devices) running on a widely dispersed set of hosts, and allows users to query those data efficiently. IrisLog is deployed on PlanetLab [22], a research platform composed of hundreds of networked nodes on three continents. Using a web-based form, users can query sets of PlanetLab nodes (*e.g.,* all the nodes at CMU) by particular metrics (*e.g.,* CPU load) and time periods over which the data have been collected (*e.g.,* last one hour). The web user interface also allows the user to issue arbitrary XPATH queries over the global XML database. In addition to the web-based front end, there are also programmatic mechanisms to query the database, so that other PlanetLab services can use IrisLog.



Deployed PlanetLab nodes.

Coastal Imaging

In collaboration with oceanographers of the Argus [14] project at Oregon State University, we have deployed a coastal imaging service on IrisNet. The service uses cameras installed at sites along the Oregon coast line and processes the live feed from the cameras to identify the visible signatures of nearshore phenomena like riptides, sand-bar formations, *etc.* For example, the figure below shows how time-averaged exposures of camera images, generated by merging the raw frames on the left, can be used to identify sand-bar formation. The front-end of this service allows users to 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 on certain events (*e.g.,* darkness).



(a) Raw video frames

(b) 10 minute time exposure

may use a database of individual identifying information (name, SSN, *etc.*) and location. Supporting the high update rates that such services may generate is quite challenging. For example, consider a person locator service that updates user positions once every ten minutes. For a moderately sized metropolitan area of 1.5 million people, the database of user locations would need to handle approximately 25,000 updates per second. For the entire United States (approximately 300 million people) there would be approximately 500,000 updates per second. One strategy for supporting these update rates is to partition the database across multiple nodes. However, such partitioning of a database often limits the types of queries supported or the efficiency of performing queries. Fortunately, while all sensing applications allow users to query the collected sensor readings, their design often targets particular types of queries. For example, a Parking Space Finder service may require availability queries to include a target location. Similarly, a person locator service query may require queries to specify a person's name and SSN. Based on these requirements, IrisNet helps developers by providing tools to implement a hierarchically organized distributed database. The IrisNet system handles the task of mapping database fragments onto different nodes and routing queries to them.

The overriding goal in providing the above support for data collection and database processing is to reduce drastically the difficulty of authoring a wide-area sensing service. In IrisNet, a developer has to do little more than write the code that performs the filtering of sensor readings and the schema that defines the contents of the database. In Section 4, we describe the details of our implementation for this support.

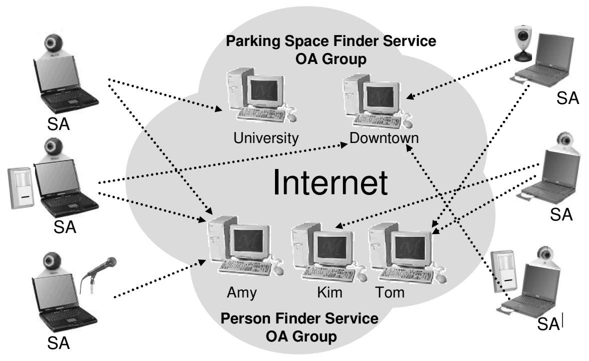


Figure 1: IrisNet Architecture

4 The IrisNet Architecture

Figure 1 shows IrisNet's basic two-tier architecture. This design is motivated by the following observations:

- Despite the differences between types of sensors, developers need a generic data acquisition interface for accessing sensors. In IrisNet, the nodes that implement this interface are called *Sensing Agents* (SA).

```
<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>
```

Figure 2: Part of the XML schema used in the Parking Space Finder service.

- Services need to store the service-specific data produced by the SAs in a distributed database. In IrisNet, The nodes that implement this distributed database are called *Organizing Agents* (OAs).

In the following subsections, we describe the architecture of OAs and SAs in detail.

4.1 OA Architecture

Service developers deploy sensing services by orchestrating a group of OAs dedicated to the service. As a result, each OA participates in only one sensing service (a single physical machine may run multiple OAs). The group of OAs for a single service is responsible for collecting and organizing sensor data in order to answer the particular class of queries relevant to the service (*e.g.,* queries about parking spots for a Parking Space Finder service). OAs also should provide fault tolerance and balance load across the system.

The Choice of Database. We envision a rich and evolving set of data types, aggregate fields, *etc.,* within a service and across services, best captured by self-describing tags. In addition, each sensor takes readings from a geographic location, so it is natural to organize sensor data into a geographic/political-boundary hierarchy. Hence, we choose

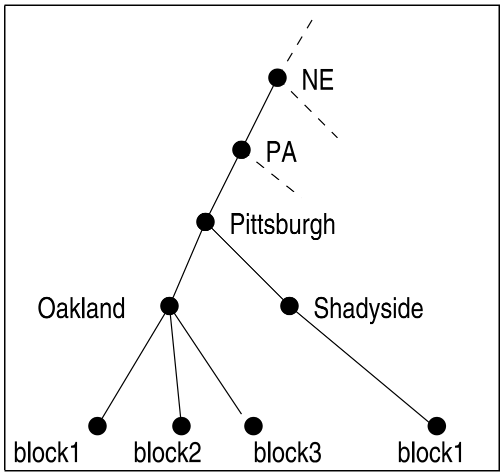


Figure 3: The hierarchy defined by the `id`-tagged nodes in the Parking Space Finder service.

to represent sensor-derived data in XML since it is well-suited to representing such data. Figure 2 shows part of an XML document describing the schema used for the Parking Space Finder service. The schema describes the static (*e.g.,* `<handicapped>`) and dynamic (*e.g.,* `<available>`) metadata as well as the hierarchy used by the service. The database schema provided by the service author identifies the hierarchical portion of the database through the use of special `id` tags. Figure 3 shows the tree representation of the hierarchy formed by the `id`-tagged nodes.

Distributing the Database. In order 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 non-contiguous subsets).

The path from the root of the hierarchy to a node defines its globally unique name (this requires the `id` tags of the sibling nodes to be unique). For example, the `Pittsburgh` node in Figure 3 is given by `city-Pittsburgh.state-PA.usRegion-NE`. Each OA registers with DNS each split node that it owns, where the registered name is the split node's global name appended with the name of the service and a registered domain suffix (*e.g.,* intel-iris.net). This is the only mapping from the logical hierarchy to physical IP addresses in the system, enabling considerable flexibility in mapping nodes in the document to OAs, and OAs to physical machines. For example, figure 4 shows an example configuration where the nodes `NE`, `PA`, and `Pittsburgh` are owned by one OA and the rest of the nodes are owned by one OA each.

Query Routing. We use the XPATH query language, because it is the most widely used query language for XML, with good query processing support. Figure 4 shows an

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

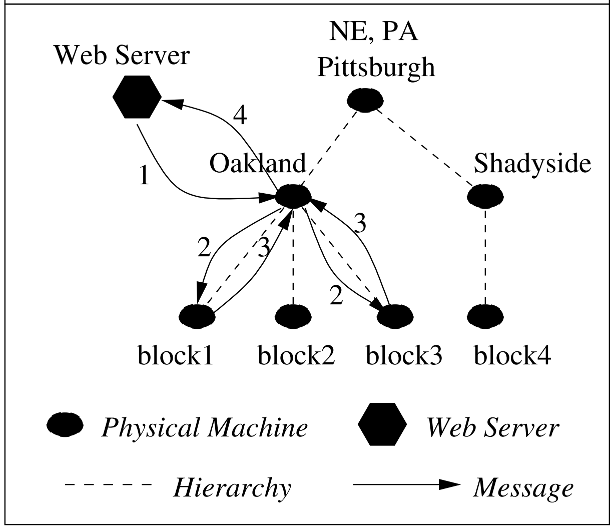


Figure 4: Top: An XPATH query. Bottom: A mapping of logical nodes to seven machines, and the messages sent to answer the query (numbers depict their relative order).

example XPATH query asking for all the available parking spaces at `block1` and `block3` of `Oakland`.

Because our XML database is distributed, providing fast and correct answers to user queries is quite challenging. The system must route a query directly to the nodes of interest to that query, and must pass data between OAs only as needed. We now show how IrisNet accomplishes these tasks.

An XPATH query selects data from a set of nodes in the hierarchy. In IrisNet, the query is routed directly to the lowest common ancestor (LCA) of the nodes potentially selected by the query. For example, `Oakland` is the LCA node for the XPATH query shown in Figure 4. Note that the globally unique name of the LCA node can be derived simply by parsing the XPATH query. A DNS lookup on this name provides the IP address of the current owner of the starting point OA. This mechanism prevents the root of the hierarchy from becoming a bottleneck; the LCA is typically far down in the hierarchy.

Upon receiving a query, the starting point OA queries its portion of the overall XML document and evaluates the result. For many queries, a single OA may not have enough of the document to respond to the query. The OA determines which part of a user's query can be answered from the local document and where to gather the missing parts (extracting the needed global names from the document). The OA looks up the IP addresses of the other OAs to contact and sends subqueries to them. These OAs may, in turn, perform a similar gathering task. Finally, the starting point OA collects the different responses and the combined result is sent back to the user. For the example in Figure 4, the `Oakland` OA receives the query from the web server, sends subqueries to the Block 1 and Block 3 OAs, who each return a list of available parking spaces, to be combined at the `Oakland` OA and returned to the user.

Caching and Data Consistency. In order to improve the performance of repeated requests for similar information (*e.g.*, multiple users requesting parking spaces downtown), OAs cache data from any query gathering task that they perform. The query processing mechanism of IrisNet uses cached data even if the new query only partially matches them. Using this cached information proves challenging since since it complicates the OA's task of identifying what remaining information must be gathered to answer a query.

Because of network propagation delays and the use of cached data, answers returned to users may not reflect the most recent data. A query may specify consistency criteria indicating its tolerance for stale data. We store timestamps along with the cached data that indicate when the data were created, so that an XPATH query specifying a tolerance is automatically routed to data of appropriate freshness.

The complete details of IrisNet's database distribution, query processing and caching appear in [8].

Fault Tolerance. IrisNet replicates nodes in the logical hierarchy on multiple OAs. IrisNet uses two types of replicas: *primary replicas* that are kept strongly mutually consistent and placed in geographically optimal locations (*e.g.*, near the sources of the sensor readings) 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, the lists of all the primary and secondary replicas are retrieved by DNS lookups, and replicas are tried sequentially until the query is successfully routed to a live OA.

4.2 SA Architecture

SAs collect raw sensor data from a number of sensors (possibly of different types). The types of sensors can range from webcams and microphones to temperature and pressure gauges. The focus of our design is on sensors such as webcams that produce large volumes of data and can be used by a variety of services. One SA can be shared by multiple services.

Figure 5 shows the execution 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.

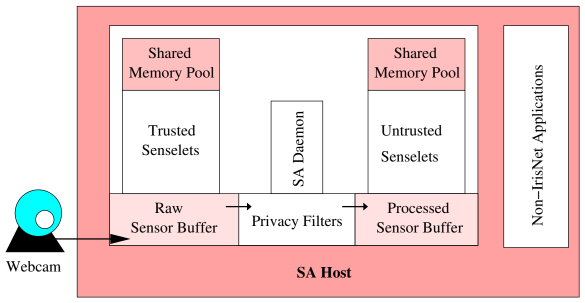


Figure 5: Execution environment in SA host

Programmable SAs. IrisNet enables services to dynamically upload and control the execution of code that filters sensor readings in a service-specific fashion. We refer to this code as a *senselet*. A senselet instructs the SA to take the raw sensor feed, perform a specified set of processing steps (as required by the specific service), and send the resulting distilled information to an OA specified by the service. A single SA may execute one or more senselets for each service that desires access to its sensor feeds.

Protection of Senselets and SA Host. The SA host and senselets are protected from buggy or malicious senselets. IrisNet runs senselets as different processes, and thus provide process-level protection among senselets. We currently assume trust between the administrator of a sensor host where senselets execute and the author(s) of those senselets, and thus trust that senselets don't consume excessive resources or exploit security vulnerabilities on the SA host. This model makes sense in cases where a single organizational entity administers the sensor hosts, and has a formal trust relationship with the senselet authors. IrisNet also supports sandboxing the senselets by executing them in a virtual machine (VM), to limit the resources a senselet can consume and the operations it can execute.

Privacy Mechanisms. IrisNet aims to provide mechanisms for service developers to implement appropriate privacy policies for their services. Our current prototype of SAs uses privacy filters to identify the sensitive components (*e.g.,* human faces) in an image, and replaces these regions with black rectangles. As shown in Figure 5, IrisNet distinguishes between *trusted* and *untrusted* senselets; trusted senselets receive raw video feeds, whereas untrusted ones only receive the privacy-filtered video data.

Shared Computation Among Senselets. The filtering done by the senselets is computationally expensive. To increase the efficiency of execution of multiple senselets on the same SA, we exploit the fact that one sensor feed may be of interest to multiple different IrisNet services. For example, a video feed in a particular location may be used in one service to monitor parking spaces, and in another to track passersby in the same visual field. Image processing primitives (*e.g.,* color-to-gray conversion, noise reduction, edge detection, maintaining a statistical model of the background, *etc.*) are reused heavily across senselets working on the same video stream. IrisNet provides a mechanism for senselets to share intermediate computational results with other senselets through a shared memory pool. Intermediate results are named using the ordered series of processing steps performed on them. In this way, a senselet can query for intermediate results that match those it needs, and avoid recomputing those results that have been computed previously by another senselet.

5 Prototype Services

Here, we present three services from very different application domains built using IrisNet. While we give overviews of these applications in the sidebars, we now describe their implementation and deployment.

Parking Space Finder. Our current prototype operates on a set of mock parking lots and miniature cars laid out on a tabletop. However, the rest of the system operates as it would in an outdoor setting. Senselets that recognize parking space availability run on laptops with webcams viewing these mock parking lots. After initial calibration of the reference background image, each senselet detects the presence of cars by comparing the current image of the spaces and corresponding background image [24]. These image processing tasks are done using the Intel Open Source Computer Vision (OpenCV) [1] library. Once a senselet determines which parking spaces are empty, it sends the availability information to the appropriate OAs.

Figure 3 shows the database schema for this application. As mentioned earlier, this schema defines the geographically hierarchical organization of the database and the dynamic (*e.g.,* availability information from the SAs) and static (*e.g.,* meter restrictions on the space) descriptions for each of the spaces.

Network and Host Monitor (IrisLog). Our current prototype of IrisLog runs on PlanetLab [22], an open, shared, planetary-scale application testbed consisting of 102 nodes distributed across 42 sites in three continents: North America, Europe, and Australia. IrisLog supports a superset of the queries supported by the currently deployed Ganglia [20] PlanetLab monitoring service, but incurs far less network overhead. Each PlanetLab node runs an SA, which uses the output of the local Ganglia daemon 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 schema for IrisLog describes the metrics to be monitored on each PlanetLab node (*e.g.,* CPU and memory load,

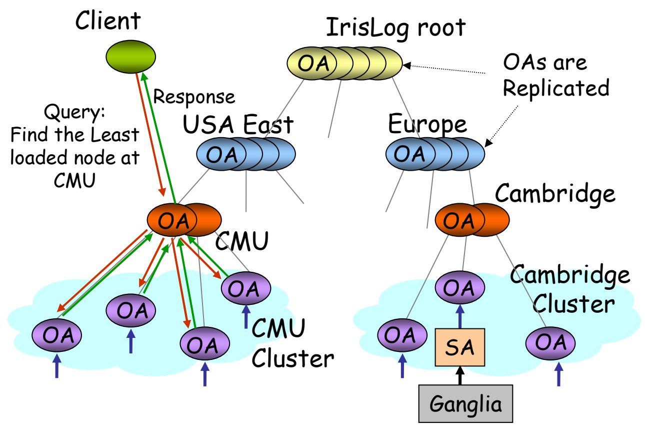


Figure 6: Hierarchy used in the IrisLog service.

bandwidth usage, *etc.*) and organizes these metrics into a geographical hierarchy. This hierarchy, part of which is shown in Figure 6, allows efficient processing of geographically scoped queries (*e.g.,* find the least loaded CMU node). To support simple historical queries over the monitored data, the schema uses multi-resolution vectors to store each monitored metric. These vectors provide higher resolution samples of recent data than older data.

A Coastal Imaging Service. Researchers from Oregon State University placed cameras and computers along the Oregon coast to monitor the ocean. The senselet for this service produces image data such as 10-minute interval snapshots, 10-minute time exposure images that show wave dissipation patterns (hence submerged sand-bars), variance images, and photogrammetric meta-data.

The schema defines a shallow hierarchy consisting of a root node with each of the coastal camera locations connected as leaf nodes. Each leaf node stores only the most recent snaphot 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 OSU and the leaf OAs are located at the corresponding coastal locations.

6 Conclusions

Sensor network research has largely focused on localized deployments of low-power wireless sensors collecting numerical measurements. However, as illustrated by the examples presented in this article, many applications require a wide-area network of powerful sensors such as video cameras. In this article, we have discussed the desirable features of a pervasive architecture for Internet-scale sensing, and the technical challenges in realizing these goals. We described IrisNet, a general-purpose software infrastructure designed for Internet-scale sensing services. We have discussed features in IrisNet that greatly simplify many common tasks in these services, such as collecting, filtering and combining sensor feeds, and enabling distributed queries with reasonable response times.

Today, we are far from the vision of highly-available, high-performance, easy-to-use Internet-scale sensing. In addition to the technical challenges discussed in this article, there are important policy, privacy, and security concerns that must be addressed before rich sensors can be deployed 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, non-profits and individuals will provide the needed sensor infrastructure.

Acknowledgements. We thank A. Deshpande and R. Sukthankar for their important contributions to the IrisNet system. We thank M. Satyanarayanan for many valuable suggestions and C. Helfrich for helping with the experimental setup.

References

[1] Intel Open Source Computer Vision Library. http://www.intel.com/research/mrl/research/opencv/.

[2] Libra: Scalable advanced network services based on coordinated active components. http://www-2.cs.cmu.edu/afs/cs.cmu.edu/project/cmcl/www/team/.

[3] D. Agrawal and S. Sengupta. Modular synchronization in distributed, multi-version databases: Version control and concurrency control. *IEEE TKDE*, 1993.

[4] R. Alonso, D. Barbara, and H. Garcia-Molina. Data caching issues in an information retrieval system. *ACM TODS*, 1990.

[5] P. Bonnet, J. E. Gehrke, and P. Seshadri. Towards sensor database systems. In *MDM*, 2001.

[6] S. W. Chen and C. Pu. A structural classification of integrated replica control mechanisms. *ACM TODS*, 1992.

[7] R. Collins, A. Lipton, H. Fujiyoshi, and T. Kanade. Algorithms for cooperative multisensor surveillance. *Proceedings of the IEEE*, 89(10):1456–1477, Oct. 2001.

[8] A. Deshpande, S. Nath, P. B. Gibbons, and S. Seshan. Cache-and-query for wide area sensor databases. In *SIGMOD*, 2003.

[9] I. Foster and C. Kesselman. *The Grid: Blueprint for a New Computing Infrastructure*. Morgan Kaufmann, 1999.

Related Work

There have been a variety of related efforts addressing a issues related to enabling Internet-scale sensing services. These can be classified into efforts that have 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 making real world measurements accessible by applications. The work on sensor networks has largely concentrated on the use of "motes," small nodes containing a simple processor, a little memory, a wireless network connection and a sensing device. Because of the emphasis on resource-constrained motes, earlier key contributions have been in the areas of tiny operating systems [13] and low-power network protocols [16]. Mote-based systems have relied on techniques such as directed diffusion [12] to direct sensor readings to interested parties or long-running queries [5] 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 [17, 18, 19], to address the limitations of sensor motes. None of this work considers sensor networks with intelligent sensor nodes, high-bit-rate sensor feeds, and global scale.

Video Surveillance. The use of video sensors has been explored by efforts such as the Video Surveillance and Monitoring (VSAM) [7] project. Efforts in this area have concentrated on image processing challenges such as identifying and tracking moving objects within a camera's field of vision. These efforts are complementary to our focus on wide-area scaling and service authorship tools.

Internet Services Frameworks. A number of different efforts, *e.g.*, [29, 27, 2, 9] have developed frameworks for simplifying the development of scalable, robust Internet services. In general, these projects target a lower level of the architecture than IrisNet. They concentrate on issues that are generic to all Internet services, such as load balancing, resource allocation and network placement. In contrast, IrisNet addresses issues that are unique to services that need to 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. The distributed database infrastructure in IrisNet shares much in common with a variety of large-scale distributed databases. For example, DNS [21] relies on a distributed database that uses a hierarchy based on the structure of host names, in order to support name-to-address mapping. LDAP [28] addresses some of DNS's limitations by enabling richer standardized naming using hierarchically organized values and attributes. However, it still supports only a relatively restrictive querying model. Harren *et al.* [11] have investigated peer-to-peer databases that provide a richer querying model than the exact-match queries supported by existing DHT systems, to limited success (significant hotspots in storage, processing, and routing were reported). DHT-based databases are more robust than the replicated hierarchically-based approach we propose, but 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 are a well-studied topic [26], with the focus on supporting distributed transactions or other consistency guarantees (*c.f.* [25, 10, 23, 3, 15, 4, 6]). None of the previous work addresses the difficulties in distributed query processing over an XML document. There is also considerable work on query processing over a federation of heterogeneous databases [26], dealing with issues such as incompatible schemas. These issues do not arise in our current architecture, because the service author defines the schema for an entire service.

[10] J. Gray, P. Helland, P. O'Neil, and D. Shasha. The dangers of replication and a solution. In *SIGMOD*, 1996.

[11] M. Harren, J. Hellerstein, R. Huebsch, B. Loo, S. Shenker, and I. Stoica. Complex queries in DHT-based peer-to-peer networks. In *IPTPS*, 2001.

[12] J. Heidemann et al. Building efficient wireless sensor networks with low-level naming. In *SOSP*, 2001.

[13] J. Hill, R. Szewczyk, A. Woo, S. Hollar, D. Culler, and K. Pister. System architecture directions for network sensors. In *ASPLOS*, 2000.

[14] R. Holman, J. Stanley, and T. Ozkan-Haller. The application of video sensor networks to the study of nearshore oceanography. submitted to *Pervasive Computing* special issue on sensor networks, 2003.

[15] N. Krishnakumar and A. Bernstein. Bounded ignorance in replicated systems. In *PODS*, 1991.

[16] J. Kulik, W. Rabiner, and H. Balakrishnan. Adaptive Protocols for Information Dissemination in Wireless Sensor Networks . In *MOBICOM*, 1999.

[17] S. Madden and M. J. Franklin. Fjording the stream: An architecture for queries over streaming sensor data. In *ICDE*, 2002.

[18] S. Madden, M. J. Franklin, J. M. Hellerstein, and W. Hong. Tag: A tiny aggregation service for ad hoc sensor networks. In *OSDI*, 2002.

[19] S. Madden, M. J. Franklin, J. M. Hellerstein, and W. Hong. The design of an acquisitional query processor for sensor networks. In *SIGMOD*, 2003.

[20] M. L. Massie, B. N. Chun, and D. E. Culler. The ganglia distributed monitoring system: Design, implementation, and experience. Submitted for publication, February 2003, http://berkeley.intel-research.net/bnc/.

[21] P. V. Mockapetris and K. J. Dunlap. Development of the Domain Name System. In *SIGCOMM*, 1988.

[22] L. Peterson, T. Anderson, D. Culler, and T. Roscoe. A blueprint for introducing disruptive technology into the internet. In *Hotnets-I*, 2002.

[23] C. Pu and A. Leff. Replica control in distributed system: An asynchronous approach. In *SIGMOD*, 1991.

[24] Shapiro and Woods. *Computer Vision*. Prentice-Hall, 2001.

[25] J. Sidell, J. Sidell, P. M. Aoki, S. Barr, A. Sah, C. Staelin, M. Stonebraker, and A. Yu. Data replication in mariposa. In *ICDE*, 1996.

[26] A. Silberschatz, H. F. Korth, and S. Sudarshan. *Database Systems Concepts*. McGraw Hill, 2002.

[27] J. R. von Behren, E. Brewer, N. Borisov, M. Chen, M. Welsh, J. MacDonald, J. Lau, S. Gribble, and D. Culler. Ninja: A framework for network services. In *Proceedings of 2002 USENIX Annual Technical Conference*, 2002.

[28] M. Wahl, T. Howes, and S. Kille. Lightweight Directory Access Protocol (v3). Technical report, IETF, 1997. RFC 2251.

[29] M. Welsh, D. E. Culler, and E. A. Brewer. SEDA: An architecture for well-conditioned, scalable internet services. In *SOSP*, pages 230–243, 2001.