IrisNet: An Architecture for Internet-scale Sensing Services

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

[*]Intel Research Pittsburgh [†]Carnegie Mellon University [‡]U.C. Berkeley

Abstract

We demonstrate the design and an early prototype of *IrisNet* (*Internet-scale Resource-Intensive Sensor Network services*), a common, scalable networked infrastructure for deploying wide area sensing services. IrisNet is a potentially global network of smart sensing nodes, with webcams or other monitoring devices, and organizing nodes that provide the means to query recent and historical sensor-based data. IrisNet exploits the fact that high-volume sensor feeds are typically attached to devices with significant computing power and storage, and running a standard operating system. It uses aggressive filtering, smart query routing, and semantic caching to dramatically reduce network bandwidth utilization and improve query response times, as we demonstrate.

Our demo will present two services built on IrisNet, from two very different application domains. The first one, a *parking space finder*, utilizes *webcams* that monitor parking spaces to answer queries such as the availability of parking spaces near a user's destination. The second one, a *distributed infrastructure monitor*, uses measurement tools installed in individual nodes of a large distributed infrastructure to answer queries such as average network bandwidth usage of a set of nodes.

1 Introduction

Imagine driving towards a destination in a busy metropolitan area. While stopped at a traffic light, you query your PDA 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. Hours later, you realize that your meter is about to run out. You query your PDA to discover that, historically, meter enforcers are not likely to pass by your car in the next hour. A half hour later, you return to your car and discover that although it has not been ticketed, it has been dented! Querying your PDA, you get back images showing how your car was dented and by whom.

This scenario demonstrates the potential utility of sensor-based services such as a Parking Space Finder, Silent (Accident) Witness and Meter Enforcement Tracker. While several research projects [1, 5, 6, 7] have begun to explore using and querying networked collections of sensors, these systems have targetted the use of closely co-located resource-constrained sensor "motes" [3, 4]. In this demo, we demonstrate an early prototype of a sensor network system architecture, called *IrisNet* (*Internet-scale Resource-Intensive Sensor Network services*) based on much more intelligent participants. We envision an environment where different nodes (standard PCs, laptops and PDAs) on the Internet have attached sensing devices. Together, these nodes form an Internet-scale collection of sensors, from webcams collecting live video to network monitors collecting real-time traffic measurements. Sensing services can retrieve information from this collection of sensors and provide service to users.

While webcams and other smart sensors are often inexpensive and easy to deploy across a wide area, realizing useful services requires addressing a number of challenges including preventing the transfer of large data feeds across the network, efficiently discovering relevant data among the distributed collection of sensor nodes, efficiently handling static meta-data information (e.g., parking meter details and map directions) and multiple sensor feeds, etc. Our goal in IrisNet is to create a common, scalable software infrastructure that allows services to address these challenges in a manageable fashion. This would enable rapid development and deployment of distributed services over a worldwide network of sensor feeds.

IrisNet is composed of a potentially global collection of Sensing Agents (SAs) and Organizing Agents (OAs). SAs collect and process data from their attached webcams or other sensors, while OAs provide facilities for querying recent and historical sensor data.

Proceedings of the 29th VLDB Conference, Berlin, Germany, 2003

Any Internet connected, PC-class device can play the role of an OA. Less capable PDA-class devices can act as SAs. Key features of IrisNet include:

- IrisNet provides simple APIs for orchestrating the SAs and OAs to collect, collaboratively process and archive sensor data while minimizing network data transfers.
- The user is presented with a logical view of the data as a single XML document, while physically the data is fragmented across any number of host nodes (*data transparency*).
- IrisNet supports the entire unordered fragment of XPATH 1.0, a standard XML query language, for querying the data in the system.
- IrisNet handles issues of query routing, semantic caching of responses and load balancing in a scalable manner for all services.

We believe that IrisNet can enable a wealth of new sensor-based services. Examples include providing live virtual tours of cities, answering queries about the waiting time at different restaurants, unobtrusive monitoring of your children playing in the neighborhood, witnessing whose dog pooped on your lawn, and determining where an umbrella was left behind.

2 The IrisNet Architecture

In this section we briefly describe the overall architecture of IrisNet, its query processing features, and its caching and data consistency mechanisms.

Architecture. IrisNet is composed of a dynamic collection of SAs and OAs. Nodes in the Internet participate as hosts for SAs and OAs by downloading and running IrisNet modules. Sensor-based services are deployed by orchestrating a group of OAs dedicated to the service. These OAs are responsible for collecting and organizing the sensor data in a fashion that allows for a particular class of queries to be answered (e.g., queries about parking spaces). The OAs index, archive, aggregate, mine and cache data from the SAs to build a system-wide distributed database for a service.

In contrast, SAs are shared by all services. An SA collects raw sensor data from a number of (possibly different types of) sensors. The types of sensors can range from webcams and microphones to network traffic monitors. The focus of our design is on sensors that produce large volumes of data and require sophisticated processing, such as webcams. The sensor data is copied into a shared memory segment on the SA, for use by any number of sensor-based services.

OAs upload service-specific scripting code (called a *senselet*) to any SA collecting sensor data of interest to the service, basically telling the SA to take its raw sensor feed, perform the specified processing steps, and send the distilled information to the OA. For video feeds, the senselet consists primarily of calls to an image processing library. (IrisNet uses the OpenCV library.) Filtering data at the SAs prevents flooding the network with high bandwidth video feeds and is crucial to the scalability of the system.

IrisNet also provides mechanisms for different services to share computations among senselets. For example, if both the Parking Space Finder service and the Silent Accident Witness service perform common steps of filtering for motion detection and for identifying vehicles, SAs automatically detect them and enable one service to use the intermediate results computed by another service.

Further details on the IrisNet architecture can be found in [8].

Query Processing. Central to IrisNet is distributed query processing. We here describe its key features briefly, and refer the readers to [2] for more details. IrisNet stores data in XML databases associated with each OA. We envision a rich and evolving set of data types, aggregate fields, etc., best captured by self-describing tags – hence XML was a natural choice. Larger objects such as video frames are stored outside the XML databases; this enables inter-service sharing, as well as more efficient image and query processing.

Data for a particular service is organized hierarchically, with each OA owning a part of the hierarchy. An OA may also cache data owned by other OAs. A common hierarchy for OAs is geographic, because each sensor feed is fundamentally tied to a particular geographic location.[1] XML is well-suited to organizing data hierarchically.

A user's query, represented in the XPATH language, selects data from a set of nodes in the hierarchy. We exploit the hierarchical nature of the OA organization to expedite the routing of queries to the data. Observe that each XPATH query contains a (maximal) hierarchical prefix, which specifies a single path from the root of the hierarchy to the lowest common ancestor (LCA) of the nodes potentially selected by the query. For a query posed anywhere in the Internet, IrisNet constructs a DNS-style name from its hierarchical prefix and performs a DNS lookup (in a DNS server hierarchy identical to the OA hierarchy) to determine the IP address of the OA (the *starting point OA*) that owns the LCA node. The query is routed directly to that OA. This reduces the response time and avoids having the root of the hierarchy become a bottleneck.

Upon receiving a query, the starting point OA queries its local database and cache, and evaluates the result. If necessary, it gathers missing data by sending subqueries to its children OAs, who may recursively query their children, and so on. Finally the answers from the children are combined and the result is sent

[1] A service may define indices based on non-geographic hierarchies too.

Figure 1: Webcams monitoring toy parking lots

back to the user. Note that the children IP addresses are found using the same DNS-style approach, with most lookups being served by the local host. The key technical challenge to overcome in our approach is how to efficiently and correctly detect, for general XPATH queries, what parts of a query answer are missing from the local database, and where to find the missing parts.

XPATH queries supported. In our current prototype, we take the common approach of viewing an XML document as *unordered*, in that we ignore any ordering based solely on the linearization of the hierarchy into a sequential document. We support the entire *unordered* fragment of XPATH 1.0, ignoring the few operators such as *position*() or axes like *following-siblings* that are inappropriate for unordered data.

Partial-Match Caching and Data Consistency. An OA may cache query results from other OAs. Subsequent queries may use this cached data, even if the new query is only a partial match for the original query.

IrisNet allows queries to specify a consistency criteria indicating a tolerance for stale data. It stores timestamps along with the data, so that an XPATH query specifying a tolerance is automatically routed to the data of appropriate freshness. In particular, each query will take advantage of cached data only if the data is sufficiently fresh.

3 A Parking Space Finder Service

The first example service that we demonstrate is that of a parking space finder. This service utilizes webcams that are monitoring parking spaces to gather information about the availibility of the parking spaces.

Sensing Agents. We use four cameras that are monitoring toy parking spaces set up as part of our demo (Figure 1). These cameras are attached to laptop machines that process the video feed, and perform image processing to decide whether a parking spot is full or empty. The setup simulates four parking lots near Intel Research Pittsburgh.

Organizing Agents. Figure 2 shows the part of the hierarchy that is used in this demonstration. This logical hierarchy is mapped onto seven OAs running on seven PCs in CMU and Intel Research Pittsburgh, as follows: (1) the four blocks corresponding to the parking lots are mapped onto one OA each, (2) the two neighborhoods, Oakland and Shadyside, are mapped onto one OA each, and (3) the remaining nodes in the hierarchy are mapped onto one OA.

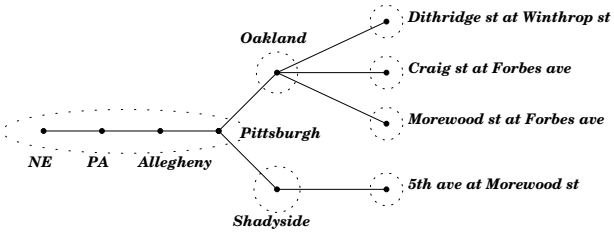

Figure 2: The hierarchy used in the demonstration and the mapping of the hierarchy onto the OAs

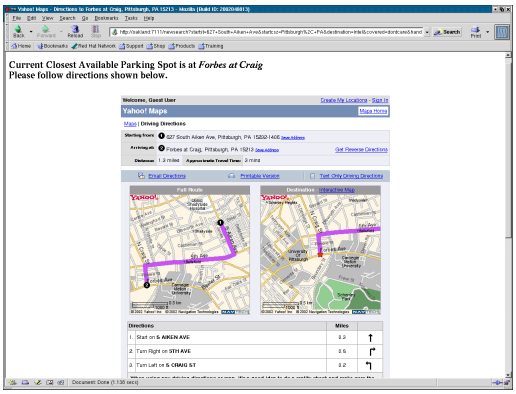

Figure 3: Driving directions to the parking spot are displayed using Yahoo Maps (maps.yahoo.com)

Web Frontend. The web frontend for this service essentially presents the user with a form that she can fill out to specify her destination, and also other constraints that she might have (*e.g.*, that the parking spot must be covered). Currently, we only allow the user to pick from five destinations near the parking lots using a drop-down menu. Once the user specifies her criteria and submits the query, the frontend finds the nearest available parking spot that satisfies the user's constraints using IrisNet, and then uses the Yahoo Maps Service to find driving directions to that parking space from the user's current location. These driving directions are then displayed to the user (Figure 3).

The driving directions are continously updated as the user drives towards the destination, if the availability of the parking spot changes, or if a closer parking spot satisfying the constraint is available. We envision that a car navigation system will repeatadly and periodically ask the query as the user nears the destination. Lacking that, we currently simulate such a behavior by resubmitting the query periodically and by assuming that the user has reached the next inter-

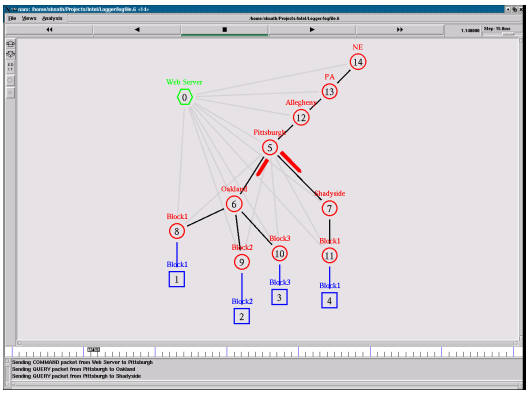

Figure 4: A modified version of NAM is used to show the messages during a query execution

section along the route.

Logging and replaying messages. We also demonstrate a mechanism that we have built for logging and replaying the messages exchanged by the web frontend and by the OAs. The collected log information during execution of a query is used to lazily replay the messages that were sent during the execution of the query. We use the NAM network simulator to show these messages. NAM is part of the popular open-source network simulator, *ns*, with a graphical display that shows the configuration of the network under consideration (Figure 4), and uses animation to show messages being communicated in the network.

A series of XPATH queries of increasing complexity are used to demonstrate visually various aspects of our system such as routing to the starting point, recursive query processing, partial-match caching, and query-based consistency.

4 A Distributed Infrastructure Monitoring Service

Our second example service, a distributed infrastructure monitor, demonstrates the usefulness of IrisNet in a different domain. The service allows users to efficiently query the current state of different nodes in an infrastructure. For the demo, we run the service on PlanetLab [9], an open, shared planetary-scale application testbed consisting of over 100 nodes distributed across dozens of sites spanning three continents: North America, Europe, and Australia.

Sensing Agents. Each PlanetLab node runs an SA. The senselet of this service is composed of two components. The first component, a *monitoring daemon*, constantly computes different statistics (*e.g.,* CPU and memory load, bandwidth usage, etc.) of the node through well-defined interfaces (*e.g.,* `/proc`, kstat, and kvm). The second component, a *publishing thread* periodically reports the statistics in XML format to the OA running on the same node.

Organizing Agents. Each node runs an OA that

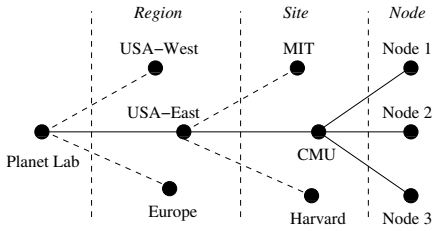

Figure 5: Part of the hierarchy used in the distributed infrastructure monitor on PlanetLab

collects the measurement statistics from the local SA and stores them in its local XML database. Additional OAs are deployed to construct a hierarchy that suits the query workload, and as backups for fault tolerance. In our demo, we use a hierarchy that organizes the OAs geographically. Figure 5 shows part of the hierarchy.

Web Frontend. In our demo we use a web frontend for this service which presents the user with a form that she can fill out to specify the set of PlanetLab nodes she is interested in (*e.g.,* all the nodes in the site *CMU*), the particular metrics she wants to consider (*e.g.,* CPU load, network bandwidth usage etc.), and the aggregate function (*e.g.,* average load of a site) to be used within the query. The frontend also allows the user to type any arbitrary XPATH query on the global XML database of the measurement data collected by the monitoring daemons.

Acknowledgement: We thank M. Satyanarayanan for many valuable suggestions.

References

[1] Bonnet, P., Gehrke, J. E., and Seshadri, P. Towards sensor database systems. In *ACM Mobile Data Management* (2001).

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

[3] Estrin, D., Govindan, R., Heidemann, J., and Kumar, S. Next century challenges: Scalable coordination in sensor networks. In *ACM MOBICOM* (1999).

[4] Kahn, J., Katz, R. H., and Pister, K. Next century challenges: Mobile networking for 'smart dust'. In *ACM MOBICOM* (1999).

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

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

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

[8] Nath, S., Ke, Y., Gibbons, P. B., Karp, B., and Seshan, S. Irisnet: Enabling sensor-enriched internet services. Intel Technical Report IRP-TR-03-04, 2003.

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