SenShare: Transforming Sensor Networks Into Multi-Application Sensing Infrastructures

Ilias Leontiadis, Christos Efstratiou, Cecilia Mascolo, Jon Crowcroft
University of Cambridge
{name.lastname}@cl.cam.ac.uk

Abstract. Sensor networks are typically purpose-built, designed to support a single running application. As the demand for applications that can harness the capabilities of a sensor-rich environment increases, and the availability of sensing infrastructure put in place to monitor various quantities soars, there are clear benefits in a model where infrastructure can be shared amongst multiple applications. This model however introduces many challenges, mainly related to the management of the communication of the same application running on different network nodes, and the isolation of applications within the network. In this work we present SenShare, a platform that attempts to address the technical challenges in transforming sensor networks into open access infrastructures capable of supporting multiple co-running applications. SenShare provides a clear decoupling between the infrastructure and the running application, building on the concept of overlay networks. Each application operates in an isolated environment consisting of an in-node hardware abstraction layer, and a dedicated overlay sensor network. We further report on the deployment of SenShare within our building, which presently supports the operation of multiple sensing applications, including office occupancy monitoring and environmental monitoring.

1 Introduction

An increasing number of sensor networks are being deployed to collect information on a wide range of applications. Relying on data from sensor networks is becoming crucial both to monitor situations such as environmental changes or business processes, but also as a means of developing context-aware ubiquitous computing applications.

It has been argued [12] that while mature networking and duty cycling protocols for sensor networks exist, approaches to sensor network sharing and management are still immature. Typical sensor networks are designed and deployed to serve a single application. Indeed, the common approach in the design of sensor networks is to deploy networks that are *fit-for-purpose* with the primary aim of supporting a single application that belongs to a single authority (usually the owner of the infrastructure) [15]. While this is a sensible approach for short-term and small-scale deployments, in sensor network deployments that consist of thousands of nodes with a life span of multiple years, inducing high costs of deployment and maintenance, the *single-application* approach can lead to inefficient use of resources and low cost-benefit results. Moreover, the requirement for dedicated sensing infrastructure to support new applications belonging to different organisations can lead to unnecessary replication of sensing infrastructure. As an example, sensors in public buildings have been used in a wide range of applications, from monitoring environmental conditions in each room [7] to supporting advanced context-aware applications [14]. In many of these applications the actual sensing modalities that are used are similar or even identical. In such environments, the 'single-application' approach employed by existing sensor networks can become counter productive. Either additional sensing infrastructure needs to be deployed to support new applications, or multiple sensing applications need to be fused into a single binary that is deployed over the network. The latter approach can be extremely complex if the end-users or authorities responsible for these applications are different.

In this work we explore a potential change of paradigm regarding the design and deployment of sensing infrastructure. Our objective is to investigate the feasibility of deploying sensing infrastructures with the primary role of allowing the sharing amongst multiple applications. The development of sensor networks designed for sharing can open new opportunities for efficient resource utilisation and adaptation to changing requirements [11]. New operational models can be envisaged, where an installed sensing infrastructure acts as a pool of sensing resources that can be harnessed by newly developed applications. In such models the cost-benefit analysis for a sensing infrastructure that is to be deployed, can involve new business opportunities that may include *leasing* the infrastructure to third parties on demand, without disrupting the operation of sensing applications already running on the network. Furthermore, we can envisage the design of sensing applications that can potentially be deployed to different networks on demand as the needs of the application owner change. We identify the domain of smart-buildings as a possible environment where shared sensing could show significant benefits. Presently, the deployment of sensing infrastructures in buildings is treated as a tool for monitoring the environmental impact of a building's life cycle and the economic impact of its maintenance [10]. However, the same sensing modalities can also be harnessed for the design of future ubiquitous computing applications, or for inferring information that have not been envisaged during the building's construction.

In this paper, we present SenShare, a system that enables sensor network sharing among different applications. Our approach is based on the design of a platform that allows the creation of multiple *virtual sensor networks* on top of a physical infrastructure. Our objective is to allow applications to operate in a virtual environment isolated from other applications running on the sensor network. This is achieved by offering a hardware abstraction layer on each sensor node which allows multiple co-located applications to use the node's hardware resources. An application can span across the whole infrastructure or a selected sub-set of the physical network. The platform supports the formation of an overlay sensor network for each application, with a dynamically constructed virtual topology that can be treated by the application as their actual topology. The SenShare platform provides a clear separation between the sensor network infrastructure and the individual applications allowing ownership and management of the infrastructure and applications by different authorities. We further report on the deployment of our system in the building of our institution. A number of sensing applications were deployed including applications for monitoring office occupancy and room environmental conditions.

2 The SenShare Platform

The challenge: In the design of SenShare we assume the presence of two user roles: (i) the infrastructure owner, and (ii) the application developers. The infrastructure owner is considered to have full control over the physical infrastructure. The application developers are assumed to have an understanding of the geography of the target environment and the sensing modalities offered by the network. The main challenge in shared sensing is to allow newly developed applications to be deployed over the infrastructure without disrupting the operation of previously installed applications. Considering the case of a smart building, one of the main reasons for the deployment of sensing technologies is to monitor the environmental conditions in the building and adjust the HVAC (Heating Ventilation Air Cooling) system accordingly. The dynamic deployment of a newly developed application that uses humidity sensors to estimate room occupancy, should operate without disrupting the pre-existing environmental monitoring sensing application. In design terms, addressing the challenge requires the support of multiple applications to operate on the same network (even co-existing on the same node), by offering protection / isolation of sensing applications both in terms of the runtime environment inside the sensor node, and the network traffic over the sensing infrastructure.

Application Isolation: The development of sensor network applications is typically designed under the assumption that the particular application is the *owner* of the physical network. This is reflected on the design of development environments like TinyOS, where the resulting application is a single binary controlling all hardware components of the sensor node. One of the design objectives of SenShare is to support TinyOS applications within a shared sensing infrastructure. The method for achieving this involves: (i) adapting the generated binary during compile time to break the tight coupling with the physical hardware, and (ii) sharing access to the hardware through a cross-application hardware abstraction layer residing on each sensor node (Figure 1). The co-execution of multiple applications on each sensor node relies on the presence of embedded operating systems that allow the dynamic loading and execution of applications during runtime. Operating systems that offer such functionality include Contiki OS for lower-end sensor nodes (e.g. TelosB, MicaZ) and embedded Linux for higher end nodes (e.g. Imote2, Gumstix).

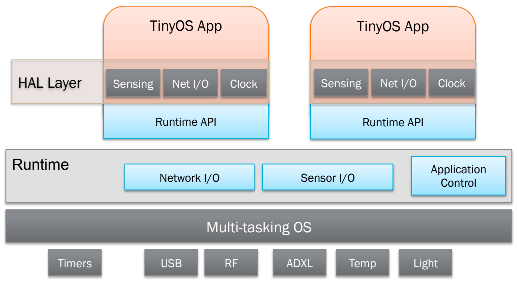


Fig. 1. SenShare Node Architecture

Compile time - TinyOS: TinyOS applications are designed to operate as single-threaded applications, with static memory management, operating directly on the hardware of the target sensor node. Furthermore, TinyOS offers a component-based development environment where applications can re-use components offering access to low-level functionality. In order to offer cross-platform compatibility, TinyOS defines three different layers of hardware abstraction. The TinyOS Hardware Abstraction Layer (HAL) defines a set of interfaces that enable the development of hardware independent applications. At compile time the appropriate low-level components are assembled to form a binary that can run on a specific platform. SenShare extends the existing TinyOS core to provide the same hardware independent functionality as other hardware platforms. However, instead of incorporating access to the physical hardware devices within the target TinyOS binary, SenShare diverts any processing for hardware requests to a runtime hardware abstraction layer that reside inside the sensor node and is shared with all running applications.

Communication between the running application and the SenShare runtime is OS specific. In embedded Linux, communication can be performed through named pipes, while in Contiki through run-time dynamic linking [2]. TinyOS supports split-phase access to hardware components, where a system call returns immediately and the completion of the operation is signalled through hardware interrupts translated into TinyOS events. Support for split-phase access is offered through an asynchronous IPC interface between the TinyOS application and the runtime layer. For a sensor access HAL component for example, a read request is sent to the runtime through the `read()` system call. The HAL component registers a callback function that is signalled from the runtime when the read is complete. The signal is then translated into a TinyOS event, thus imitating the effect of a hardware interrupt.

In order to offer support for TinyOS, we implemented a new target platform within the TinyOS toolchain (v.2.1.0). The SenShare TinyOS platform implements abstractions for common TinyOS components. Specifically, it includes components for sensor access, LEDs, timers, (read-only access to) RTC, IEEE 802.15.4 interface, and UART (USB) access. TinyOS applications that are compiled for the SenShare platform incorporate these virtual hardware controllers that are responsible for passing hardware access requests to the SenShare runtime.

Runtime: The in-node environment consists of a permanent in-node component, the *Node Runtime*, that is owned and managed by the infrastructure owner, and the dynamically loaded TinyOS applications that can run inside the sensor node. The runtime operates as a separate process and builds, within each sensor node, an abstraction layer that controls access to the hardware resources of the node. This access is performed transparently through the use of the virtual hardware components that are linked to every TinyOS application compiled for the SenShare platform.

The *sensor I/O* provides an abstraction of the sensing components of the node. Access to hardware components is performed asynchronously. Requests to access a sensor are delivered from applications and when the sensor reading is complete an event is raised to trigger the corresponding application. The challenges we tried to address in the sensor I/O were to limit the overhead caused by the introduction of an additional layer of abstraction, and to care for possible race conditions caused by multiple applications requiring high sampling frequency for the same sensor. We address these challenges by introducing a *bursting* queue for handling multiple sense requests: When a request for reading is submitted by an application, if the corresponding queue is empty, the request is served immediately. Upon completion of the low-level access to the sensor, if more requests have been submitted and queued, they are all served immediately with the newly received reading. The impact of this technique can be estimated if we consider the maximum average frequency rate the runtime can deliver. If t_s is the time required to take a reading from a sensor. For n applications competing for the same sensor, the maximum average frequency rate that can be achieved is $f = \frac{1}{nt_s}$. With the *bursting queue* the sensor can satisfy together all sensing request that are submitted while a sensor read is performed. Let t_p be the processing time for returning a sensor value to a pending application. It can be shown that the maximum average frequency that can be achieved is $f = \frac{1}{t_s + nt_p}$. Considering that $t_s > t_p$ the *bursting queue* can achieve better performance than blocking sensing when $n > \frac{t_s}{t_s - t_p}$ applications run on the node.

The *network I/O* component is providing an abstraction over the network interfaces of the sensor node. The network I/O provides an asynchronous API for sending and receiving messages, while restricting low-level access to the communication device (e.g. applications cannot change the radio channel of the interface). Low-level communication attributes, such as the communication channel, or the duty cycling policy are considered part of the infrastructure configuration. The network I/O component maintains a single priority queue for all transmission requests submitted by running applications. If the sensor network is configured to employ a radio duty cycling policy, messages are queued until the next communication window. When the queue fills up, packets are dropped according to application priorities and limits over the traffic that can be generated by each application. By decoupling the duty cycling policy from the application, the platform enables the infrastructure owner to specify bounds over the radio traffic, and energy consumption committed for communication tasks, for each sensor node. In addition to the interface abstraction, the network I/O component is responsible for implementing a virtual overlay topology for each application running on the infrastructure. The details of the overlay mechanism are presented in Section 4.

The SenShare runtime has been implemented and evaluated on the Imote2 sensor nodes (Marvel PXA27x ARM processor 400MHz, 32MB FLASH, 32MB SDRAM). The target hardware was selected for its advanced features (more memory, faster CPU) allowing us the experiment more extensively with the co-presence of multiple applications on each node. Each SenShare sensor node runs embedded Linux (v.2.6.29) as its native operating system. A sandbox environment for each application was implemented by creating a "chroot jail", as supported by the `chroot()` system call in Linux. The sandbox environment restricts access to all physical devices on the node, thus ensuring that applications are only allowed to access the hardware through the SenShare runtime. The SenShare runtime is installed in all sensor nodes running as a user–space process. Running applications communicate with the runtime through named pipes.

3 Infrastructure Management

The in-node runtime environment, in addition to controlling access to hardware resources, is also responsible for maintaining a network management layer to control and maintain the overall sensor network (e.g., start/stop applications, monitor resources). SenShare uses a collection tree routing protocol (CTP) [4] to both collect messages from the network and deliver control messages to the nodes. For the requirements of SenShare we implemented a modified version of CTP that allows the delivery of messages both up and down the tree. In addition to the CTP protocol, global clock synchronization is provided by the use of the TPSN [7] protocol.

Selective Control & Deployment The application control service allows the owner of the network infrastructure to micro-manage the operation of all applications running on the infrastructure. Each application deployed over the SenShare infrastructure is assigned a unique identifier. This unique identifier is combined with an SQL-like target selection to direct commands to specific nodes over the network. The application control service can support the following commands:

- `startApp(appId, selection)`: Start a previously deployed application on selected nodes.
- `stopApp(appId, selection)`: Stop a running application on selected nodes.
- `disableDevice(appId, device, selection)`: Restrict access to a hardware device (sensor or network interface) for an application on selected nodes.

– `enableDevice(appId, device, selection)`: Allow access to a hardware device (sensor or network interface) for an application on selected nodes.

The deployment of new applications is performed in two phases: i) selection phase: using SQL-like commands target nodes are selected according to location, sensing modalities and available resources, ii) dissemination phase: the binary of the application is delivered to the target nodes, using a modified version of Deluge [5]. After the installation of a new application the SenShare platform forms a virtual overlay topology that can be perceived by the application as a dedicated physical sensor network, isolated from any other traffic. As applications can be installed on nodes further apart, such overlay topology needs to bridge distant application instances into a connected virtual topology.

4 Overlay Sensor Networks

Traffic Isolation One of the requirements of the overlay network is to isolate the network traffic of an application from any underlying mechanisms used to maintain the overlay or any traffic from other applications. In order to achieve such isolation the SenShare runtime extends each application packet with an application routing header. The application routing header is 6 bytes long and has the following format: $\{app\_id, seq\_no, origin, destination\}$
where *app_id* is the unique application id, *seq_no* is a sequence number that along with the other fields of the header is used to discard duplicates, *origin* and *destination* are the addresses of sender and receiver respectively.

Each application transmits a network message formatted according to the IEEE 802.15.4 standard. The runtime attaches the application routing header storing the original source and destination address specified by the application. On receipt of a packet the information on the header is used to reconstruct the packets source and destination address to its original values. This way packets that have been delivered over multi-hop virtual links are perceived by the application as being sent by a single-hop neighbor.

Overlay Topology During the deployment of an application on the SenShare infrastructure, application instances can be installed on any subset of nodes in the network. Some application instances can land on nodes that are physical neighbors, and therefore can communicate with single hop messages. Multiple such nodes that are in proximity to each other can form clusters where application instances are part of a connected topology of single hop neighbors.

However, in the general case an application deployment can result in a number of *clusters that are isolated from each other*. The purpose of the overlay formation is to link these isolated clusters by establishing virtual links between them allowing each application to form a single connected network. To establish these links intermediate SenShare nodes that *do not run an instance of the application* need to act as relays between these disconnected partitions.

The overlay formation problem can be defined as follows:

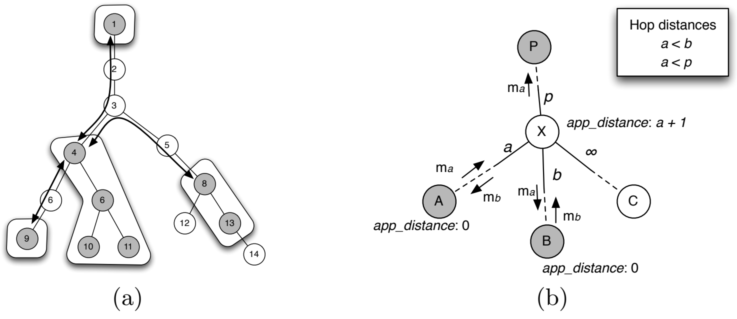


Fig. 2. Overlay formation: (a) Overlay topology on top of the existing collection tree. (b) Routing on forwarding nodes

- For each application identify the nodes that are on the edges of a connected node cluster.
- For nodes that are on the edges of clusters discover optimum paths that connect them to other clusters
- Ensure that all clusters are connected together and can access the network's sink.

The last requirement is imposed in order to ensure that all running applications can connect to the sink and deliver data to back-end applications.

One of the key features of the overlay management protocol is the reliance on the routing mechanism that is already employed by the network to support the infrastructure management. To avoid additional overhead, we exploit the routing tree in order to discover new paths that can connect disjoint application clusters. This approach resulted in a light-weight protocol for discovering the most efficient links on top of the existing tree topology that can bring all application clusters together.

Overlay Formation The overlay formation protocol utilizes the underlying CTP topology to discover routes for virtual connections between clusters. A brief description of the protocol is:

- *Identify cluster edges* Cluster edges are nodes connected to tree branches where their next hop neighbor does not run the application. For example, in Figure 2(a) nodes 1, 4, 8, 9 and 13 are cluster edges.
- *Discover distance to closest cluster* After the installation of a new application each node measures their distance to the closest cluster edge lower down the routing tree. In the example, node 3 has a 1 hop distance to the closest cluster while node 4 has 0.
- *Connect edges to closest cluster* Packets from cluster edges are routed to the closest cluster edge with smaller tree depth than their own. In this way, lower clusters are connected to clusters higher on the tree until they are all connected to the sink (see Figure 2(a)).

Implementation Each node on the network maintains two tables:

- `app_distance`: Holds the distance to the closest node running a specific application lower down the tree. The table size is O(a), where a is the number of running applications.
- `neighbor_distance`: Holds the distances of the next-hop neighbors (children and parent) over the tree. The table size is O($a * c$), where a is the number of running applications and c is the number of children.

When an application starts on a given node it sets its local `app_distance` to 0 and informs its neighbors. When a node receives such a message from a child it updates its local distance with the minimum value, and stores the neighbor's distances on `neighbor_distance`. If there is a change on its local distance the node will propagate that information to its neighbors. In the example on Figure 2(a) node 8 reports a distance of 0, node 5 updates its distance to 1 and notifies 3. Node 3 will not update its distance as it has a closer cluster reported from 4 and would not propagate any changes over the tree. Distance updates are performed when applications are installed and when a node changes parent (due to failure or congestion).

Using these two tables each node is responsible for routing messages between the closest clusters. The routing policy employed on each node is:

```
message_received_from_node(A)
if (local_distance == 0) // We run the application
    accept_message
else if (distance(A) > (local_distance - 1)
    send_to_closest_neighbor
else // A is the closest neighbor
    send_to_all_links(x) with distance(x) > distance(A)
    send_to_parent
```

The purpose of the routing policy is, for each application instance, to establish communication with a set of neighbors (virtual or real) that are running the same application. For nodes that are within an application cluster, neighbors are their one-hop physical neighbors. In these cases the protocol does not require to perform any additional routing. The routing policy is responsible for forwarding messages between cluster edges in order to establish virtual neighbors. This forwarding is performed by nodes that do not run the application. Figure 2(b) shows an example of this routing mechanism. Node X is connected with clusters of distance a, b and p with $a < b$ and $a < p$. Messages arriving from the B cluster are delivered to A as this is the closest cluster. Messages from A are sent to all clusters with longer distances and to the node's parent. In this way cluster B is connected to A, and A is connected to both B and P.

5 Evaluation

In this section we present our results about i) the application isolation penalty, ii) the network overlays. Furthermore, we present our observation and results from our own deployment.

Runtime performance penalty. We evaluate the overhead of the SenShare runtime by comparing the performance of applications operating as native sensing applications without the use of the SenShare platform, against the performance with the platform. We particularly looked at the impact in terms of sampling rates, CPU and memory load. We evaluated the runtime's overhead by running two types of applications in our system: i) an application that tries to sample a sensor (temperature) as frequent as possible and ii) an application that samples a number of sensors at a rate of 10Hz. For each one of these applications we implemented two versions: a *native* version that directly accesses the hardware using native OS system calls, and the *virtualized* one that accesses the sensors using the SenShare API.

As expected, the results show an overhead due to the extra layers that mediate between the hardware and the application. Figure 3(a) shows the maximum sampling rate that an application can achieve through the abstraction layer versus an application that is running alone on the node. The results show that the maximum sampling rate is reduced by 28% due to the delays added by the abstraction layer. However, when more than one application run simultaneously, our optimization mechanisms that combine requests from multiple applications into one hardware access, reduce the impact of the platform and offer comparable performance. These results illustrate one case where in-node coordination of multiple applications can result in improved overall system performance.

To investigate the performance in less demanding conditions, we experimented with an application that can detect the occupancy of a desk by a desk-mounted sensor, monitoring desk vibrations at a rate of 10Hz. In Figures 3(b) and 3(c) we plot the CPU and memory usage of the native and the virtualized version (the virtualized version includes the overhead of the application and the SenShare runtime). As expected, we observe that there is an overhead due to the additional CPU and memory resources consumed by the SenShare runtime, compared to the native applications without the runtime support. However, in both cases the overhead is linear and therefore grows relatively slowly with respect to the number of applications deployed: this is the price we pay for the sharing of the infrastructure and it is mitigated by the ability to use it by different parties at the same time.

Network Overlay. In order to evaluate the performance of the overlay routing protocol we set up a testbed of 25 sensor nodes running the SenShare platform (Figure 4(a)). The nodes where arranged on a grid formation of 5x5. For the particular testbed setup, the transmission signal strength on the radios was reduced in order to allow the creation of more realistic topologies. In this setup each node was able to communicate only with nodes directly next to each other.

For this evaluation we deployed a number of applications on randomly selected nodes over the grid. The experiments were grouped in terms of the size of the deployment: 5, 10, 20, 25 nodes. Each test application was designed to produce periodic beacons messages every 500 milliseconds that were broadcast to all their virtual neighbors. For all the experiments we collected topology information on the formed overlay networks as well as network traffic statistics.

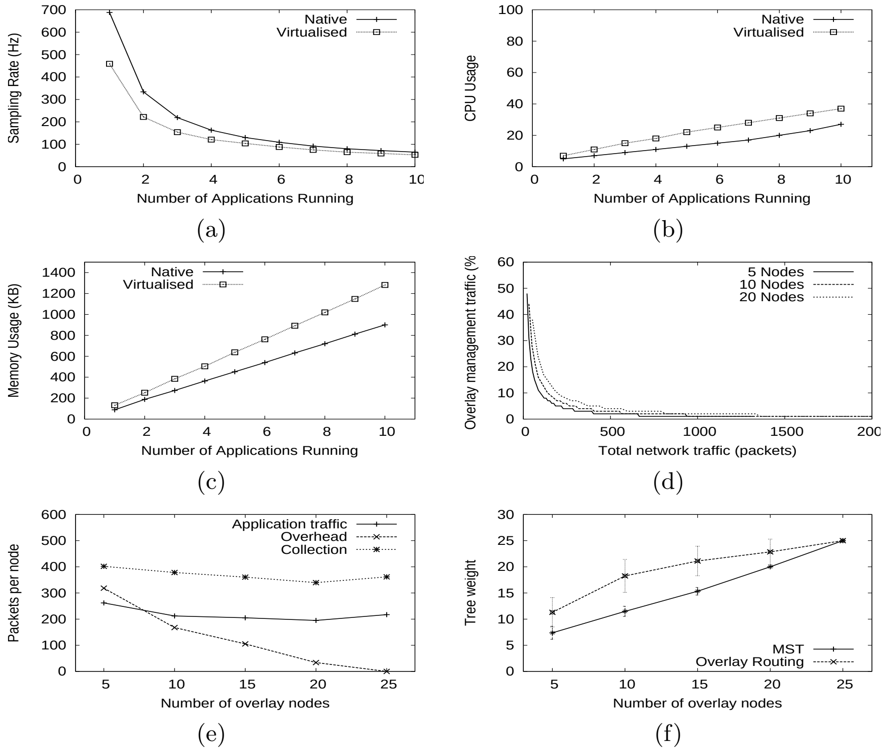
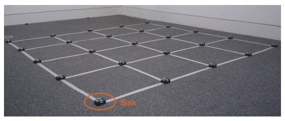
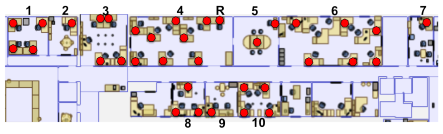


Fig. 3. Evaluation results. Runtime performance comparing Native vs. Virtualized applications: (a) Maximum sampling rate, (b) CPU usage, (c) Memory Usage. Overlay performance: (d) Percentage of traffic for maintaining the overlay versus total network traffic, (e) Overlay routing overhead vs overhead of a collection routing, (f) Tree weight of topology created by the Overlay routing vs. a minimum spanning tree.

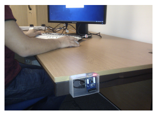
Overlay Management Cost: One important metric for the evaluation of an overlay protocol is the relative cost required to maintaining the topology against the actual *useful* traffic generated by the application. A key feature of the SenShare overlay protocol is the low overhead in traffic that is necessary to maintain an overlay topology. Figure 3(d) shows the percentage of the overlay management traffic versus the traffic generated by the applications in the experiments. As seen from the results, the impact of the overlay management traffic is diminishing over time to less that 0.1 percent. In fact, after the initial exchange of distance vectors between the nodes participating in an overlay, the overlay management would only be triggered to exchange messages if a node changes their parent. During our experiments with overlays of 20 nodes, in some occasions the CTP triggered a change of parent due to congestion. However, even in these cases where the network traffic was high enough to trigger a change of parent, the overlay management traffic was negligible (less that 3 packets on average for



(a) Overlay evaluation testbed.



(b) Deployment topology: A node was attached to every desk. 'R' indicates the location of the sink.



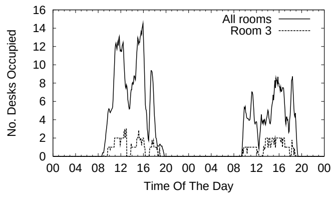
(c) The desk-mote

Fig. 4. The SenShare deployment

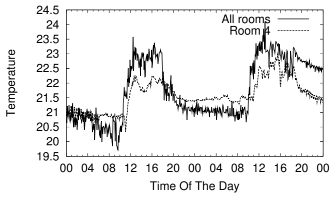
each parent change). This high performance in terms of network overhead is only achieved because the overlay management service relies on the CTP protocol to maintain the underlying infrastructure. However, as such a routing mechanism is assumed in many real deployments, the low impact overlay protocol is an attractive option.

Overlay Traffic Cost: In order to evaluate the total overhead of the overlay mechanism, we compared the traffic generated by the nodes running the applications, versus the traffic of routing nodes that were relaying packets between disjointed overlay clusters. For comparative reasons we compared this overhead with a simple collection mechanism where each application packet was relayed to the sink to maintain global connectivity. The results are shown in Figure 3(e). As expected the overhead for small size deployments, where target nodes can be further apart, is higher. However, for deployments where nodes form larger clusters, the overhead drops significantly. In the case of 25 nodes, the overlay mechanism is effectively inactive as all nodes are part of a connected cluster. Moreover, the overlay mechanism offers significant gains over a collection mechanism that could be used directly over CTP. The gain is achieved as the SenShare overlay forward traffic only to the closest overlay cluster avoiding unnecessary tree floodings.

Overlay Topology: In order to evaluate the effectiveness of the overlay protocol, we calculated the total weight of the produced overlay tree (a link's weight is calculated based on the expected number of transmissions) and compared it with a calculated minimum spanning tree (MST) connecting all application nodes, as an indication of the optimal topology. As shown in Figure 3(f), the total weight of the overlay topology varies significantly in our experiments. This variation is caused by the different topologies of the underlying tree that can affect the performance of the overlay topology. Compared to the MST, the SenShare protocol does not always produce the optimal overlay topology, although the overlay performs better as the number of deployed nodes increases. In the design of any such protocol there is a clear trade-off that needs to be considered between the high cost of discovering and maintaining the optimal graph and the resulting gains in performance. SenShare offers a light–weight protocol for establishing overlay networks with minimal impact in maintaining the overlay topology.



(a) Office occupancy



(b) Temperature measurements

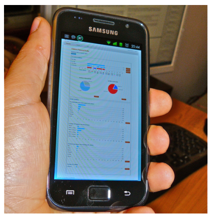

(c) The web front-end

Fig. 5. SenShare deployment

6 Deployment

The SenShare platform has been successfully deployed in the office space of our research institution. The platform has been operating for more than three months supporting two primary applications, with additional experimental applications deployed for shorter periods of time and with more applications being scheduled for the near future. The deployment consists of 35 iMote2 nodes equipped with a sensor board that supports temperature, humidity, ambient/infrared light, and acceleration sensing. Each node is connected to permanent power source and is mounted on every desk in an area of 10 offices (Figure 4(b)). Furthermore, additional sensors were attached to various appliances, such as coffee machines in kitchens and communal areas. A number of applications were designed as typical sensor network applications that would operate on a dedicated network, and were implemented in TinyOS.

Room Environmental Conditions: This was the first application that was deployed over the shared infrastructure. The environmental monitoring application is responsible for recording temperature and humidity levels for all ten offices where the SenShare platform was deployed. The application is installed on one node per room. Figure 5(b) shows a snapshot of the data collected over the period of two days. Certain rooms such as room 4 exhibits lower temperatures in average. This can be explained by the fact that these rooms have a larger window surface and dissipate more heat.

Office Occupancy: The second application that we deployed is responsible for monitoring people that are currently working in every office that is instrumented with sensors. The application utilizes the accelerometer on each node to monitor the vibrations on the desk. Applying the appropriate local filtering of events and a hysteresis mechanism to smooth out irregular vibrations, the application can accurately record the time that people spent at their desk and send the appropriate events over the overlay network. Figure 5(a) shows a snapshot of the collected data for two days of the week (Thursday and Friday). Overall occupancy can change during the day due to certain global events such as lunch breaks (around 12.00) or during a popular seminar (Thursday at 16.00).

Appliance Monitoring: Specialised applications were deployed for the sensors attached to appliances such as coffee machines. These applications used various sensing modalities to detect operation patterns of the device. The collected events were reported back to the deployment's root node.

The SenShare deployment is still a dynamic environment where experimental applications are deployed by different developers without disrupting pre-existing sensing applications. Indeed the open availability of the infrastructure has triggered a range of new applications that take advantage of the sensing resources. An example is shown in Figure 5(c) where a web-based application was written to visualise the collected data (i.e., the users can receive both real-time and historical information about their time at work, find out if there is fresh coffee or if someone is currently using the kitchen, etc).

Experiences
Since the deployment of SenShare, a number of developers had the opportunity to experiment with the infrastructure and deploy their sensing applications. The following paragraphs summarise some of the experiences we had working with a shared sensing environment.

Support Tools: The complexity of an environment where multiple stakeholders were simultaneously developing and testing sensing applications underlines the importance of management and deployment tools. In our experience, operating in this environment, the automated deployment and reprogramming of the network proved to be essential. Furthermore, the ability to use SenShare's routing protocols to both collect debug messages and remotely control individual applications, were key tools for application testing. Through this experience we cannot emphasise enough the importance of such mechanisms in the development of sensing applications, a fact that is often neglected in the literature.

Opportunistic network installation: The fact that SenShare relies on the same routing mechanism irrespective of the actual application running on each node proved extremely useful for the progressive deployment of the infrastructure. During the early stages of the deployment, the infrastructure covered only the office spaces of the building. When later we attached sensor nodes to appliances in kitchens and common rooms, the nodes were seamlessly incorporated in the existing network, although these sensor nodes were typically running different applications. Over time the SenShare installation is slowly transforming into a "patchwork" where sensing devices supporting completely different applications form a common sensor network allowing the routing of all messages between application instances. The overlay network allows the transparent passing of messages over network segments that serve different purposes.

7 Related work

In supporting the deployment of hardware independent applications over a sensor network [1], in-node virtualization has been suggested as a possible approach to abstract from the sensor node hardware. A trend in providing virtualization for lower-end sensor nodes relies on the design of specialized bytecode interpreters that reside inside each sensor node and offer a fine-grained control over the execution of applications. A prime example of this is Maté [6]. In the design of SenShare we follow a minimalist approach, relying on the virtualization of only specific hardware components and avoiding the need for a bytecode interpreter. Indeed, SenShare applications are executed as native applications within the host OS. Furthermore, by not relying on specialized bytecode, existing applications developed using common tools, such as TinyOS can be re-compiled to run over SenShare without additional effort. Nevertheless, as the hardware abstractions offered by SenShare are designed to offer advanced features, such as infrastructure management and overlay networks, we consider them complementary to the runtime support provided by existing virtualization frameworks. The design of the SenShare runtime is well suited to be incorporated to such frameworks and give them access to these advanced features. An alternative approach for allowing sharing of sensing infrastructures is by configuring a network to deliver events in response to varying triggering conditions (utilizing an SQL-like API [8] for example), possibly satisfying the needs of multiple applications. This is a feasible approach for certain classes of applications where an SQL-like API is enough to capture the required information. However, a wide range of applications require the deployment of application specific code on the sensor nodes that is able to detect specific patterns on the low level sensor values and generate appropriate events[3,13]. SenShare aims to offer support for such applications where specific sensing functionality on the sensor node is required.

There have been important efforts in the area of micro-kernels supporting multiprocessing on low end sensor nodes [9]. We consider the work on SenShare complementary to these systems. The particular implementation presented in this work is running on embedded linux, although on smaller sensor nodes multiprocessing real-time kernels could offer the ability to run multiple tasks.

8 Conclusions

Sensor networks are generally purpose-built and single-owned infrastructures. As the demand for sensed information both for business and recreational purposes is soaring, we believe that there is a clear need for a new paradigm in the design of sensing infrastructures. Shared sensing has the potential to allow the rapid implementation and deployment of applications that can harness the sensing resources that are available in the environment. In this paper we illustrate the feasibility of building and deploying such infrastructure in a real environment. SenShare provides shared sensing using an in-node hardware abstraction layer to allow multiple applications to operate on each sensor node, and an overlay management protocol of dynamically formulated overlay topologies for each running application. Clearly, given the nature of a shared network supporting applications developed by different users, significant and possibly novel security measures must be taken to protect a network. In our future work, we intend to investigate the new classes of threats that can arise in such environments and expand SenShare with additional security and management mechanisms.

Acknowledgments: We would like to acknowledge the support of the EPSRC through project FRESNEL EP/G069557/1. We also thank all the members of the NetOS group for their support and feedback.

References

1. S. Bhattacharya, A. Saifullah, C. Lu, and G.-C. Roman. Multi-application deployment in shared sensor networks based on quality of monitoring. In *Proceedings of Real-Time and Embedded Technology and Applications Symposium (RTAS)*, pages 259–268, Stockholm, Sweden, April 2010.
2. A. Dunkels, N. Finne, J. Eriksson, and T. Voigt. Run-time dynamic linking for reprogramming wireless sensor networks. In *Proceedings of the Fourth ACM Conference on Embedded Networked Sensor Systems (SenSys 2006), Boulder, Colorado, USA*, November 2006.
3. C. Fischione, A. Speranzon, K. H. Johansson, and A. Sangiovanni-Vincentelli. Peer-to-peer estimation over wireless sensor networks via lipschitz optimization. In *Proceedings of the* 8^{th} *ACM/IEEE International Conference on Information Processing in Sensor Netwokrs (IPSN 2009), San Francisco, USA*, April 2009.
4. O. Gnawali, R. Fonseca, K. Jamieson, D. Moss, and P. Levis. Collection tree protocol. In *SenSys '09: Proceedings of the 7th ACM Conference on Embedded Networked Sensor Systems*, pages 1–14, New York, NY, USA, 2009. ACM.
5. J. W. Hui and D. Culler. The dynamic behavior of a data dissemination protocol for network programming at scale. In *Proceedings of the 2nd international conference on Embedded networked sensor systems*, SenSys '04, pages 81–94, New York, NY, USA, 2004. ACM.
6. P. Levis and D. Culler. Mate: A tiny virtual machine for sensor networks. In *International Conference on Architectural Support for Programming Languages and Operating Systems, San Jose, CA, USA*, Oct 2002.
7. J. Lu, T. Sookoor, G. Ge, V. Srinivasan, B. Holben, J. Stankovic, E. Field, and K. Whitehouse. The smart thermostat: Using wireless sensors to save energy in homes. In *8th ACM Conference on Embedded Networked Sensor Systems (SenSys 2010)*, November 2010.
8. S. R. Madden, M. J. Franklin, J. M. Hellerstein, and W. Hong. Tinydb: an acquisitional query processing system for sensor networks. *ACM Trans. Database Syst.*, 30:122–173, March 2005.
9. R. Mangharam, A. Rowe, and R. Rajkumar. Firefly: a cross-layer platform for real-time embedded wireless networks. *Real-Time Systems*, 37:183–231, 2007.
10. A. Marchiori and Q. Han. Distributed wireless control for building energy management. In *2nd ACM Workshop On Embedded Sensing Systems For Energy-Efficiency In Buildings, Zurich, Switzerland*, nov.-2 2010.
11. J. Steffan, L. Fiege, M. Cilia, and A. Buchmann. Towards multi-purpose wireless sensor networks. In *Proceedings of Systems Communications, 2005*, pages 336 – 341, aug. 2005.
12. M. Welsh. The next decade of sensor networking. Keynote speech, 7^{th} European Conference on Wireless Sensor Networks, Coimbra, Portugal, 2010.
13. G. Wittenburg, N. Dziengel, C. Wartenburger, and J. Schiller. A system for distributed event detection in wireless sensor networks. In *Proceedings of the* 9^{th} *ACM/IEEE International Conference on Information Processing in Sensor Netwokrs (IPSN 2010), Stockholm, Sweden*, April 2010.
14. A. Wood, G. Virone, T. Doan, Q. Cao, L. Selavo, Y. Wu, L. Fang, Z. He, S. Lin, and J. Stankovic. ALARM-NET: Wireless sensor networks for assisted-living and residential monitoring. *University of Virginia Technical Report*, 2006.
15. J. Yick, B. Mukherjee, and D. Ghosal. Wireless sensor network survey. *Computer Networks*, 52(12):2292–2230, August 2008.