Towards a Layered Architecture for Object-Based Execution in Wide-Area Deeply Embedded Computing

Tarek Abdelzaher, Qing Cao, Raghu Ganti, Dan Henriksson, Maifi Khan, Jin Heo, Chengdu Huang, Praveen Jayachandran, Hieu Khac Le, Liqian Luo, Yu-En Tsai

Department of Computer Science
University of Illinois at Urbana Champaign
Urbana, IL 61801

Abstract

Sensor networks introduce a new application domain and set of challenges in distributed computing including new network-level programming languages, global system abstractions, and general-purpose communication protocols. These challenges are brought about by the tight integration of computation, communication, and distributed real-time interaction with the physical world. With the growing interest in interconnecting different sensor networks across a wide-area communication infrastructure, an overarching challenge becomes one of arriving at an agreed-upon global sensor network architecture that ensures interoperability. Unlike the Internet, where a layered communication stack (namely, the TCP/IP stack) defines the network architecture, a sensor network architecture must unify not only communication interfaces but also programming interfaces, since network communication and computation functions are tightly intertwined. In that sense, the sensor network architecture refers to a layered stack of distributed computing abstractions. This paper presents an architecture and key considerations in designing and interconnecting local and global sensor networks. Candidate protocols and middleware instantiations are described from the authors' ongoing work that meet the discussed considerations.

1. Introduction

Investigation of programming tools and communication abstractions that simplify sensor network application development is a key challenge in deeply embedded systems. This challenge is amplified by the trend towards interconnecting multiple sensing systems across wide-area networks. The lack of an agreed-upon architecture for interconnecting sensors and sensor networks significantly complicates the development effort. Developers have to cope with the diversity in user requirements, diversity in sensing modalities (heterogeneity), tight integration of computation, communication, and real-time interaction with physical world, system scale, data management, and possibly disconnected operation.

Development of future commercial network software should be accessible to the average programmer, who is not necessarily a computer scientist. Towards that end, a transformational change is needed in programming languages, network architecture, and operating system support that achieves the next quantum leap in sensor networks evolution and exports the technology at scale to the broader market of scientific, industrial, and government applications. This paper describes a candidate architecture that provides operating system, network, and language support to reduce the sensor network development effort.

At present, no agreed-upon architecture exists for the sensor network protocol stack. Communication is typically lumped into system support for higher-level abstractions geared for the application programmer. Historically, approaches to reduce the sensor network development effort have primarily relied on high-level programming paradigms to make development easier. The multitude of proposed paradigms and abstractions include node-based [10], group-based [26, 25], event-based [4, 5, 14], database-centric [18, 27, 19, 20], state-centric [15], and virtual-machine-based approaches [12]. Despite the above advances, current sensor network programming practice remains very cumbersome and inefficient using low level languages such as NesC [8].

The plethora of mostly-unused high-level programming paradigms in current sensor network literature reflects the fact that it is very hard to choose the right level of programming abstraction to support the diversity of sensor network applications. Many currently proposed programming languages (and underlying communication support) have been inspired

Proceedings of the 10th IEEE International Symposium on
Object and Component-Oriented Real-Time Distributed Computing (ISORC'07)
0-7695-2765-5/07 $20.00 © 2007 IEEE

primarily by one application model, making the number of such languages of the order of the number of currently deployed applications.

For programming support to be viable, it needs to meet two fundamental goals. The first goal is *generality*. Many current high-level language prototypes fall short of this goal. For example, the authors' own Envirotrack [3] is a high-level language developed primarily for target tracking applications. It simplifies programming by abstracting away the tracking algorithms. State-centric programming [15] is motivated by the same set of applications, but caters to the statistician or control engineer who wants to design precisely the tracking algorithms Envirotrack purports to abstract away. TinyDB [20] allows the user to send queries into the network, which is great for exporting sensor data views, but less useful for developing in-network data processing protocols. With one language per application, broad support for the language is unlikely to emerge, keeping development cost high and re-use low.

The second goal in the design of our programming framework is *abstraction layering*. A consequence of generality is that we do not want to constrain all programmers to a single level of abstraction. We need to support both the end user and the low-level protocol developer. Hence, programming interfaces must be provided at different layers. The goal of the programming interfaces should lie in separating concerns between layers at different degrees of abstraction, while providing access to each (as opposed to exporting a single abstraction layer that hides all details beneath it into one black box). The idea is similar to the way a network protocol stack separates concerns by layering different protocols. In this context, current high-level programming systems for sensor networks can be described as single-layer systems. They export one level of abstraction to the programmer. No additional layers partition that code into separate concerns at lower levels of abstraction.

Finally, from a networking perspective, the purpose of sensor networks is information distillation. The network transforms a large volume of data bits to a smaller amount of useful information to users. This is a different goal from point-to-point communication that motivates the traditional Internet protocol stack. It is a higher-level function that relies both on communication and in-network computation. Consequently, many fundamental network services, protocols, and abstractions must be re-assessed.

In the following, we present an architecture and set of design considerations that stem from the aforementioned challenges. This architecture includes abstractions for programming, communication, and resource access. The abstractions are layered and together constitute a framework into which sensor network applications, protocols, and communication components can be interoperably plugged-in to compose large sensing systems. From a programming perspective, the abstraction hierarchy must start at the single node level, where the local operating system provides local sensor access and one-hop communication interfaces for the low-level programmer. We argue that a transport layer is needed that is the thin waist of the architecture and provides group-level communication APIs. This layer separates network concerns from application-development concerns. It helps implement event-centric and data-centric programming paradigms. Above that layer, different class libraries implement the abstraction of objects of different granularity (e.g., nodes and node sets). This layer supports group-based programming paradigms. By facilitating the implementation of node-centric, data-centric, event-centric, and group centric abstractions our layered architecture comes closest to the goal of generality compared to prior programming support.

The rest of this paper is organized as follows. Section 2 describes the fundamental sensor node functionality that comprises the lowest layer in the architecture. Section 3 describes the transport layer. It implements an n-to-1 communication protocol which is the main building block for higher-level abstractions. It reflects the fundamental fact that sensor networks are deployed primarily for data collection and distillation (i.e., reduction of data to information), which motivates the n-to-1 abstraction. Section 4 discusses the distributed object layer that sits above our n-to-1 communication. This layer exports remotely accessible objects that encapsulate distributed state making it easily accessible via appropriate method invocations. Section 5 presents challenges and architectural support for addressing disconnected operation. Section 6 takes a closer look at wide-area network issues and extensions. The paper concludes with Section 8, which summarizes the architecture and presents directions for future work.

2. The Sensor Node Layer

Our architecture for sensor networks is a hybrid of a programming abstraction and a communication protocol hierarchy. Since objects offer a flexible way of layering different abstractions, we choose an object-based framework in which different layers are implemented as different object classes. Implementations of these classes form libraries that can be made available to the developers. The lowest layer in that hierarchy is the sensor node layer.

To support objects at the bottom of our architecture, we develop a light-weight operating system, called LiteOS, that provides the basic object compilation and execution environment. It presents a

Proceedings of the 10th IEEE International Symposium on
Object and Component-Oriented Real-Time Distributed Computing (ISORC'07)
0-7695-2765-5/07 $20.00 © 2007 IEEE

simple uniform interface to sensors (sensor objects) and communication that can be implemented on low-end sensor nodes and higher-end devices alike. This interface defines the sensor layer.

Single nodes in a sensor network provide two main functions that require agreement on interface. These functions are (i) access to local sensing resources and (ii) access to local communication resources. The interface to sensing resources is best implemented as one of a simple I/O device. Following a UNIX model, sensors can accept a write() and read() call. The former call sets up control parameters such as physical sampling frequency, sensor resolution (e.g., for multimedia sensors), and polling versus event-driven data retrieval mode. The latter call reads (buffered) sensory measurements. Hence, to the layers above, sensors are similar to files written to by the environment.

The second basic interface provided by individual nodes is one hop communication. The layer exports objects of class radio with methods such as send(), receive() and neighborhood(), which perform local communication or return a set of handles to a node's neighbors as discovered by the underlying communication protocols such as SP [6]. Programming languages such as Kairos [10] propose logical nodes as a main language abstraction. Node-based abstractions can be easily built on top of the above classes.

A simplified C++ compiler that we developed for the motes enables object-oriented programming on sensor nodes and takes advantage of those object classes exported by the operating system. The C++ language is chosen because of its widely known syntax and its support for objects. Overhead is kept low by disallowing certain dynamic features of C++, hence, allowing compile-time optimizations to efficiently reduce the code footprint and memory consumption. The primary feature supported by our C++ compiler for motes is the concept of classes, which we consider important to the object-based high-level abstractions needed. Another important feature is basic inheritance that facilitates object declaration. LiteOS provides object-based encapsulation of local operating system resources. It exports a set of C++ compatible class libraries for access to both hardware resources, such as clock, flash and LEDs, and software resources, such as timers and files. These are provided in addition to the sensor and radio interfaces discussed above.

Observe that previous efforts in the embedded community have lead to other C++ compilers that implement subsets of the language. A well-known example is *embedded* C++ [1]. However, this language is too heavy-weight to fit within the memory resources of a mote.

Another feature of LiteOS is that it replaces the rather error-prone and often counter-intuitive event-based model that is used by popular sensor network operating systems such as TinyOS and T2 [13]. Instead, LiteOS allows the user to write multi-threaded applications, and provides pre-emptive and prioritized scheduling between different threads. Our preliminary experiments demonstrate that increased memory consumption, due to the preemptive multi-threaded kernel employed by LiteOS, is acceptable even on sensor hardware platforms with the most constrained memory (4K bytes on Mica2 and MicaZ).

LiteOS, together with its C++ compatible object interface to node resources, will provide the foundation for higher-level layers described in subsequent sections. Next, we elaborate the fundamental building block that is the thin waist of our communication architecture.

3. The Transport and Aggregation Protocol

In prior work, network and application protocols where often inextricably (vertically) integrated making it difficult, for example, for different distributed applications to coexist and communicate in the network, unless they are written using the same high-level paradigm. An abstraction layer that separates network and application concerns allows diverse applications to communicate without restricting the application-level programming paradigm. We propose that this layer be the thin waist of the communication protocol stack. We call it the sensor network *transport and aggregation protocol* (TAP). This layer does not commit to any particular network addressing format. That decision depends on the routing layer. The transport layer may lie on top of multiple alternative routing layers. When an application generates a message, a header field specifies the addressing format. This field informs the transport protocol which routing protocol to use underneath. Due to the central role of the transport protocol as the waist of our architectural hourglass, the abstractions of the TAP layer and their justification are described next.

3.1 TAP Abstractions

Communication in sensor networks tends to have a many-to-one pattern. This is a natural consequence of data distillation (reduction of large volumes of data to a smaller amount of actionable information) as a main network function. Hence, the transport protocol exports many-to-one connection ports as the main abstraction. Since in-network aggregation and other processing are often performed on sensor nodes, the transport layer must execute on all nodes, performing hop-by-hop (as opposed to end-to-end) functions. Most importantly, the transport layer must support a large variety of sensor network programming paradigms. Many-to-one ports are quite useful in that regard. First, they can be used in

Proceedings of the 10th IEEE International Symposium on
Object and Component-Oriented Real-Time Distributed Computing (ISORC'07)
0-7695-2765-5/07 $20.00 © 2007 IEEE

event-based programming paradigms (in the style of SensorWare [4] and TinyGALS [5]), as well as database-centric programming paradigms (such as Directed Diffusion [11] and TinyDB programming [20]), where interests or queries are disseminated first, then matching sources form a many-to-one communication pattern with the origin to send back related data. Many other programming paradigms have notions of groups of nodes such as network regions [25], neighborhoods [26], rooted spanning trees, or object tracking groups [3]. Communication from such groups can be captured naturally via many-to-one connection ports.

3.2 Many-to-One Ports

Many-to-one ports achieve generality and separation of concerns in that they uniformly express different application-defined groupings of nodes using the same API. For example, nodes that match the attributes of a given query may use the same transport port to communicate their result. Alternatively, nodes that believe they sense the same target can use the same port to communicate telemetry on that target. Observe that ports exported by TAP are different from TCP or UDP ports. While the latter are a demultiplexing index on top of a unique routing identifier, the former are intended to be a unique group identifier in a given sensor network. Uniqueness can be ensured with high probability by using a large enough field (e.g., 4 bytes) for the port number. In that sense, TAP ports are more akin to (protocol independent) multicast group IDs. While node grouping semantics differ depending on the application, the transport layer and protocols below it are isolated from such semantics, acting only on port numbers and routes among port sources and sinks.

In the authors' implementation of the TAP layer, a connection is established implicitly when one or more sources select the corresponding (same) port for data communication. The following basic semantics are associated with a connection port:

- *Filtering:* It is assumed that data originating from the same port number can be fused together. The actual fusion mechanism (if any) is application specific and is defined by a filter function specified in connection declaration. An application may define a null function indicating that aggregation is not allowed. The transport layer runs on every node in the network applying the filter where the filter is available.
- *Latency:* When a connection is declared, a latency-bound may be associated with it. All data segments originating at the corresponding port should be delivered within the specified delay bound. While hard guarantees are not given, the latency bound is used to derive flow priority accordingly.
- *Ordering*: No ordering semantics are assumed among packets originating from different source nodes of the same connection. It is generally true that data such as temperature values collected from a region do not need to be strictly ordered.

To save run-time overhead, a programmer can statically specify the filter and latency requirements for each port. These are then compiled into the network at deployment time. It is possible to statically create different port numbers (even for the same data type) for the purpose of associating different timing constraints with them. For example, it is possible to statically create two different kinds of ports namely, *normal* and *urgent* for the temperature type, each with a different time constraint. The application can use the most appropriate port depending on the desired latency.

3.3 Real-Time Response

In modern sensor networks, a sensor node may host multiple sensors of different time-constants, which imposes different constraints on freshness motivating the design of predictable-latency communication protocols. These protocols should be capable of both discrimination among flows of different urgency and enforcement of their respective end-to-end timing constraints.

In TAP, enforcement of timing behavior takes advantage of port filters to adapt the amount of data that flows through the network. For each traffic class (i.e, set of packets with same end-to-end latency constraint), we develop a filtering heuristic based on packet delay at each node. First, the desired end-to-end latency is translated into a per-hop latency requirement, T_{set}. The actual per-hop latency during overload, T_{load}, may be greater than T_{set}. The difference between the two is the latency error. We adapt the degree of aggregation for each class in a per-class feedback control loop based on the current latency error to meet the desired class latency on each hop. The degree of aggregation ultimately controls the amount of network traffic, which in turn controls network load and delay. When multiple packets are ready for transmission, an EDF policy is used to sequence them. This model assumes that the type of filtering performed on sensor data can be varied in extent. For example, an application that computes a temperature map of a region can obtain results of different degrees of fidelity by varying the amount of temporal and spatial aggregation performed on temperature data in the network. In an overloaded network, a trade-off exists between getting an approximate map quickly and getting an exact map after a longer delay. Observe that the above scheme

Proceedings of the 10th IEEE International Symposium on
Object and Component-Oriented Real-Time Distributed Computing (ISORC'07)
0-7695-2765-5/07 $20.00 © 2007 IEEE

assumes a connected network. As we show later (in Section 5), it is not always the case that the network is connected. In partitioned networks, real-time delays cannot be guaranteed. A different criterion will be used for congestion control. The transport layer protocol can be configured for real-time guarantees or for disconnected operation.

4. The Distributed Object Layer

As mentioned above, the main function of a sensor network is reduction of data to actionable information. This reduction may take one of several forms. For example, it may manifest itself in queries (conversion of large volumes of raw sensor data into concise answers to desired questions), event synthesis (conversion of large numbers of low-level events into higher-level events), or statistical filtering (conversion of raw data into higher-level statistical or aggregate values), just to name a few. Therefore, appropriate programming abstractions, at their core, must facilitate information reduction. To support a diverse set of information reduction paradigms, the main class library in our architecture is that of distributed (information reduction) objects, we call the distributedObject class. The main objective of this class library is to provide a higher-layer programming abstraction where distributed data at multiple nodes, grouped together in some paradigm-specific manner, are made accessible as state of a single centralized object with a unique identity and remote accessibility. Fundamentally, the object generally abstracts away two underlying main functions; (i) a group management function that specifies a domain or scope over which data are to be collected, and (ii) an aggregation function that specifies how the collected data are to be combined. This key abstraction is elaborated below.

4.1 The Distributed Object Class

This class exports a single core method called getState(), which returns the aggregate state of the distributed object. Programming is made easier because state can then be accessed as if it resides locally on one central machine and not distributed across the network. A large number of different programming paradigms can be implemented by defining appropriate subclasses of the distributedObject class. For example, in a database-centric paradigm, the implementation of getState() might involve querying the network. In a tracking system, getState() might return a triangulated object position. In order to define an object subclass, the programmer must define two other methods, namely, defineIdentity() and defineState(). The former implicitly defines the grouping criteria for nodes that constitute the same distributed object (which we call, node membership) in the given paradigm. Node membership can be defined, for example, by geographic location, hop count, or proximity from a target given by a specified sensory signature. The method can be thought of as a function that returns the identity of objects of the given class, if any, that a node is a member of. The function defineState() identifies the type of state to be collected from members of the distributed object. It specifies (i) sensors or local variables that supply the state, and (ii) the aggregation function to be applied on them if any.

Observe that the distributedObject class maps nicely to our many-to-one transport layer protocol support. In particular, the node membership determined from the implicit specification in defineIdentity() defines the membership of the port. Further, the aggregation functions, if any, given in defineState() give rise to filters to be used for this port type for in-network aggregation. The methods defineIdentity() and defineState(), which define an object class are elaborated below.

4.2 Abstracting Distributed Object Identity

An important function of the distributedObject class is to provide general programming support for expressing grouping semantics of nodes (into objects) in a given programming paradigm. There are two primary types of node-to-object mapping proposed in previous literature. In the first, the mapping is static. For example, in a ubiquitous computing application, a sensor node attached to a door may statically represent the door object (and export its state, such as open or closed). The defineIdentity() method is just a configuration lookup by node ID that returns the object this node represents. The second type of node grouping semantics is by region, as defined by some sensory or topological criterion. For example, distributed region objects, defined in [25], are identified by geographic location or proximity to other nodes. Functions to test for proximity and geography can be provided for programmers to use within the defineIdentity() method. In tracking applications such as Envirotrack [3], abstract objects represent targets in the environment. Nodes belonging to such an object identify themselves when they detect a sensory signature consistent with the object. A library of sensor drivers and common simple sensor processing functions may be used within the defineIdentity() method to identify proximity to different targets. This library constitutes another separate abstraction layer. Signal processing engineers, for example, can work on better functions for sensor nodes to indicate new object types (e.g., functions mentioned in [9]). Higher-level programmers can then use these to define distributed objects. The idea is to seed the class with representative

Proceedings of the 10th IEEE International Symposium on
Object and Component-Oriented Real-Time Distributed Computing (ISORC'07)
0-7695-2765-5/07 $20.00 © 2007 IEEE

functions to be extended by the community as new distributed objects and paradigms arise.

4.3 Abstracting Distributed Object State

The defineIdentity() method is called periodically by the run-time middleware on each sensor node to determine its membership of different distributed object types. Once a node determines that it is a member of a given distributed object it will need to collect and communicate the corresponding state as defined by the defineState() method. The primitive identifies (i) the local variables and sensor outputs that should be part of the distributed object state, and (ii) the aggregation functions defined on these variables to produce aggregate state. An extensible library of common aggregation functions (including functions like center of gravity) is defined, which we call aggregationClass. This class helps isolate concerns. Observe that the aggregation function is nothing but the filter mentioned earlier in describing the transport protocol. New forms of aggregate state can be supported by augmenting the aggregationClass library with new types of filters, giving other programmers more freedom in defining object state. When a node identifies itself as belonging to some object type (i.e., defineIdentity() for that type evaluates to true), it creates a many-to-one port and uses our transport layer to collect the state of that object using the variables and aggregation filters specified in defineState(). The leader maintains a view of the aggregate state that can be queried using the getState() method common to the entire distributedObject class.

5. Disruption-Tolerance Considerations

Many sensor networks operate in the field in a disconnected manner until a user collects the logged data. The disconnected model has interesting implications on the network protocol stack design. In particular, the n-to-1 ports supported by the transport protocol may no longer have a permanent physical connection between the sources and the destination. Hence, the common mode of operation should not preclude sources and destination from residing in different network partitions. The underlying routing protocol must buffer data at intermediate hops until they can be forwarded (when different partitions temporarily merge). Since buffers may become full, congestion control at the transport layer should no longer concern itself only with avoiding link over-utilization. Instead, an integral function of congestion control becomes that of ensuring proper utilization of network *storage* as well. In the following two subsections, we discuss the implications of disconnected operation on the network and transport layers respectively.

5.1 Network Layer Considerations

The disconnected operation model assumes only sporadic contact among network partitions during network lifetime. For example, a user may choose to visit the field periodically for maintenance purposes (e.g., to remove dirt and debris that may occlude light sensor inputs over time). Such visits may be used for opportunistic data upload (and reduction). The user might carry a data mule device [22] that collects data wirelessly from encountered nodes and dumps these data later to a basestation (e.g., a computer in the user's office). A primary concern of the sensor network in this model becomes that of maximizing effective storage capacity (i.e., minimizing data loss due to flash memory overflow while the network is not connected).

The above introduces the need for coordinated network storage services as an integral part of the sensor network architecture. A storage layer for sensor network applications should facilitate disconnected operation. This layer should buffer data when the network is partitioned and employ data redistribution schemes to optimize sharing of network storage when buffers become (nearly) full. The amount of data that can be stored in the network is also affected by the power consumption of nodes. Naturally, nodes will be unable to record data after energy is depleted. The storage layer should take into account the rate of energy consumption to avoid depletion-related data loss. The network storage layer is best integrated with the network layer (which, on the Internet, is responsible for routing). Indeed, it is the responsibility of the routing layer to deliver data to remote destinations. If the network is partitioned, the data should be temporarily stored. This layer should provide feedback to the transport layer on in-network storage availability. The transport layer should take this information into account in congestion control decisions.

The addition of a network storage layer (to routing) as a key architectural feature reflects a change in paradigm for sensor network operation from communication-centric to storage-centric.

5.2 Transport Layer Considerations

The interplay between storage and data reduction in wide-area networks of embedded sensors offers interesting new directions for rethinking basic transport-layer network functions such as congestion control. Both the nature of the congested resource and the actions taken to reduce congestion should be revisited. TCP congestion control implicitly assumes that the congested resource is the communication link

Proceedings of the 10th IEEE International Symposium on
Object and Component-Oriented Real-Time Distributed Computing (ISORC'07)
0-7695-2765-5/07 $20.00 © 2007 IEEE

bandwidth. In contrast, in a network whose main function is information storage and reduction, one main resource of interest is storage space. Congestion control, in this context, refers to averting not only link overload but also storage exhaustion. New degrees of freedom are offered in congestion control mechanisms for recovery from storage overload. For example, information reduction can be performed more aggressively as a congestion control mechanism to alleviate high storage utilization.

Previous solutions to the congestion control problem have generally been based on feedback mechanisms that reduce the sending rate of the source. This is attributed in large part to the point-to-point nature of connections for which such protocols are designed (e.g., TCP). In contrast to TCP flows, sensor network flows typically involve data collection from a geographically distributed area. Spatial distribution offers a unique opportunity to reduce connection traffic by consolidation in space as opposed to in time. A solution to the congestion control problem is therefore to perform adaptive data filtering in the network (a special case of which was described in the context of enforcing delay guarantees). When a multipoint-to-point connection is created, the particular filter function involved can be applied with an increased frequency as dictated by congestion control to reduce network traffic appropriately. This over-aggregation may be thought of as a form of controlled information loss that we perform to maintain proper network resource utilization or timing. The transport layer also makes decisions regarding which packets to select for fusion (by filters) based on spatial and temporal locality.

6. Globally Distributed Sensor Networks

An important challenge in sensor networks is that of writing applications that extend beyond a single sensor network patch. An application, for example, might compare temperature and humidity conditions in multiple mosquito breeding grounds around the nation, correlating them to outbreaks of the West Nile Virus. From a networking perspective, it is interesting to note that integration of myriads of mobile physical devices into the networking infrastructure fundamentally changes the main function of the (wide-area) network in several respects. Current networks are designed primarily as an infrastructure for communication between end-points. This paradigm implicitly assumes that, on average, on some global level, the collective ability of receivers to sink information is matched to the collective ability of senders to source it. This justifies abstractions, such as reliable delivery (of TCP), that dictate that every bit sent must be received.

With the constant addition of new sensing devices to the network, promoted by their decreasing cost and increasing proliferation (by Moore's Law), the total bandwidth of data production in the network can grow unboundedly. In contrast, the human capacity for consumption does not evolve as quickly. This overall imbalance creates an increasing need for information filtering and reduction as the main network function to replace simple data communication. It is this reduction that will bridge the widening bandwidth gap between networked producers and consumers. The network architecture should thus be redesigned around a new (primarily multipoint-to-point) information reduction paradigm.

Extrapolating our proposed architecture to wide-area networks; sensors, aggregators (filters), tracking objects, together with information firewalls and actuators may be implemented as edge services on the current Internet with a standard API that allows easy interconnection of these components. The information flow would cross the layer hierarchy, making several wide-area network traversals on the way among the respective edge-services. While TCP has only a myopic view of a single network traversal, protocols are needed that take into account the end-to-end picture of this information cycle. Such protocols are, in principle, wide-area extensions of TAP. For example, they can adapt the degree of data reduction at the aggregation layer according to current network conditions or balance storage at the edge of the wide-area network.

7. Conclusions

In this paper, we presented architectural considerations in the design of local and wide area sensor networks. Several key take-home points have been articulated that arise from the distinctive nature of sensor networks; in particular, their adoption of in-network computation. First, since sensor networks integrate computation and communication, a network architecture must consider distributed programming abstractions. This observation has a major impact on the architecture proposed in this paper. Object classes and their hierarchy augment traditional protocol hierarchies. Second, the fundamental network function is not communication but information distillation. This distinction brings to the forefront a different set of network services compared to those of the Internet. In particular, storage plays a much more important role in protocol design. Congestion control algorithms will need to address storage bottlenecks as well. Finally, the network is not normally connected. Hence, a disruption-tolerant approach is adopted to accommodate network partitions. Different pieces of the proposed architecture have already been implemented by the authors. A testbed is under construction to experiment with an end-to-end system that integrates reference implementations of the

Proceedings of the 10th IEEE International Symposium on
Object and Component-Oriented Real-Time Distributed Computing (ISORC'07)
0-7695-2765-5/07 $20.00 © 2007 IEEE

proposed layers. Experiences with this system are will be shared in a subsequent publication.

References

[1] Embedded c++. http://www.caravan.net/ec2plus/

[2] T-mote sky. http://www.moteiv.com.

[3] T. Abdelzaher, B. Blum, Q. Cao, D. Evans, J. George, S. George, T. He, L. Luo, S. Son, R. Stoleru, J. Stankovic, and A. Wood. Envirotrack: Towards an environmental computing paradigm for distributed sensor networks. In *ICDCS '04: Proceedings of the International Conference on Distributed Computing Systems*, 2004.

[4] A. Boulis, C.-C. Han, and M. B. Srivastava. Design and implementation of a framework for efficient and programmable
sensor networks. In *MobiSys '03: Proceedings of the 1st international conference on Mobile systems,
applications and services*, pages 187–200, New York, NY, USA, 2003. ACM Press.

[5] E. Cheong, J. Liebman, J. Liu, and F. Zhao. Tinygals: a programmingmodel for event-driven embedded systems. In *SAC '03: Proceedings of the 2003 ACM symposium on Applied computing*, pages 698–704, New York, NY, USA, 2003. ACM Press.

[6] D. Culler, P. Dutta, C. T. Ee, R. Fonseca, J. Hui, P. Levis, J. Polastre, S. Shenker, I. Stoica, G. Tolle, and J. Zhao. Towards a Sensor Network Architecture: Lowering theWaistline. In *HotOS*, 2005.

[7] R. K. Ganti, P. Jayachandran, T. F. Abdelzaher, and J. A. Stankovic. Satire: a software architecture for smart attire. In *MobiSys*, 2006.

[8] D. Gay, P. Levis, R. von Behren, M. Welsh, E. Brewer, and D. Culler. The nesc language: A holistic approach to networked embedded systems. In *PLDI '03: Proceedings of the ACM SIGPLAN 2003 conference on Programming language design and implementation*, pages 1–11, New York, NY, USA, 2003. ACM Press.

[9] L. Gu, D. Jia, P. Vicaire, T. Yan, L. Luo, A. Tirumala, Q. Cao, J. A. Stankovic, T. Abdelzaher, and B. Krogh. Lightweight detection and classification for wireless sensor networks in realistic environments. In *Sensys*, San Diego, CA, 2005.

[10] R. Gummadi, O. Gnawali, and R. Govindan. Macro-programming wireless sensor networks using kairos. In *DCoSS*, 2005.

[11] C. Intanagonwiwat, R. Govindan, and D. Estrin. Directed diffusion: a scalable and robust communication paradigm for sensor networks. In *MobiCom '00: Proceedings of the 6th annual international conference on Mobile computing and networking*, pages 56–67, New York, NY, USA, 2000. ACM Press.

[12] P. Levis and D. Culler. Mat: A tiny virtual machine for sensor networks. In *ASPLOS-X: Proceedings of the 10th International Conference on Architectural Support for Programming Languages and Operating Systems*, 2002.

[13] P. Levis, D. Gay, V. Handziski, J.-H. Hauer, B. Greenstein, M. Turon, J. Huio, K. Klues, C. Sharp, R. Szewczyk, J. Polastre, P. Buonadonnao, L. Nachman, G. Tolleo, D. Cullero, , and A. Wolisz. *T2: A Second Generation OS For Embedded Sensor Networks*. Stanford Technical Report TKN-05-007.

[14] S. Li, Y. Lin, S. H. Son, J. Stankovic, and Y. Wei. Event detection services using data service middleware in distributed sensor networks. *Telecommunication Systems, Special Issue on Information Processing in Sensor Networks*, 26(2-4), 2004.

[15] J. Liu, M. Chu, J. Liu, J. Reich, and F. Zhao. State-centric programming for sensor-actuator network systems. *Pervasive Computing, IEEE*, 2(4):50–62, 2003.

[16] J. Liu, J. Liu, J. Reich, P. Cheung, and F. Zhao. Distributed group management for track initiaition and maintenance in target localization applications. In *IPSN '03: Proceedings of the 2nd International Workshop on Information Processing in Sensor Networks*, April 2003.

[17] T. Liu, C. M. Sadler, P. Zhang, and M. Martonosi. Implementing software on resource-constrained mobile sensors: experiences with impala and zebranet. In *MobiSys*, 2004.

[18] S. Madden, M. J. Franklin, J. M. Hellerstein, and W. Hong. Tag: a tiny aggregation service for ad-hoc sensor networks. *SIGOPS Oper. Syst. Rev.*, 36(SI):131–146, 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 '03: Proceedings of the 2003 ACM SIGMOD international conference on Management of data*, pages 491–502, New York, NY, USA, 2003. ACM Press.

[20] S. R. Madden, M. Franklin, J. Hellerstein, and W. Hong. Tinydb: An acquisitional query processing system for sensor networks. *ACM Transactions on Database Systems*, 30(1),March 2005.

[21] R. Newton, Arvind, and M. Welsh. Building up to macroprogramming: An intermediate language for sensor networks. In *IPSN '05: Proceedings of the Fourth International Conference on Information Processing in Sensor Networks*, 2005.

[22] R. Shah, S. Roy, S. Jain, and W. Brunette. Data mules: Modeling a three-tier architecture for sparse sensor networks. In *SNPA*, May 2003.

[23] J. Shin, L. Guibas, and F. Zhao. A distributed algorithm for managing multi-target identities in wireless ad-hoc sensor networks. In *IPSN '03: Proceedings of the 2nd International Workshop on Information Processing in Sensor Networks*, April 2003.

[24] I. Vasilescu, K. Kotay, D. Rus, M. Dunbabin, and P. Corke. Data collection, storage, and retrieval with an underwater sensor network. In *SenSys*, 2005.

[25] M. Welsh and G. Mainland. Programming sensor networks using abstract regions. In *NSDI '04: Proceedings of the First USENIX/ACM Symposium on Networked Systems Design and Implementation*,March 2004.

[26] K. Whitehouse, C. Sharp, E. Brewer, and D. Culler. Hood: a neighborhood abstraction for sensor networks. In *MobiSys '04: Proceedings of the 2nd international conference on Mobile systems, applications, and services*, pages 99–110, New York, NY, USA, 2004. ACM Press.

[27] Y. Yao and J. Gehrke. The cougar approach to in-network query processing in sensor networks. *SIGMOD Rec.*, 31(3):9–18, 2002.

Proceedings of the 10th IEEE International Symposium on
Object and Component-Oriented Real-Time Distributed Computing (ISORC'07)
0-7695-2765-5/07 $20.00 © 2007 IEEE