# A Programmable Routing Framework for Autonomic Sensor Networks

Yu He, Cauligi S. Raghavendra
Department of Computer Science
University of Southern California
{yuhe, raghu}@usc.edu

Steven Berson, Bob Braden
Information Sciences Institute
University of Southern California
{berson, braden}@isi.edu

## Abstract

*This paper proposes a programmable routing framework that promotes the adaptivity in routing services for sensor networks. This framework includes a universal routing service and an automatic deployment service. The universal routing service allows the introduction of different services through its tunable parameters and programmable components. The deployment service completes the configuration of the universal routing service throughout a sensor network in an automatic and energy-efficient way. With this deployment service, a self-configuring ability is realized for sensor routing services. With the changeable parameters and programmable components of the universal routing service, the self-optimizing as well as other autonomic abilities can be explored in an experimental sensor network conforming to the proposed framework.*

## 1. Introduction

Technological advances have led to the emergence of small, low-cost, and low-power devices that integrate microsensing with on-board processing and wireless communication capabilities ([16], [26], and [27]). These sensors can be deployed on a large scale and deeply embedded within the physical environment. Each sensor node can operate autonomously with no central point of control and can make decisions based on the information it currently has. Networks composed of these sensor nodes can support many new applications such as physiological monitoring, environmental monitoring (air, water, soil, chemistry), precision agriculture, transportation, factory instrumentation and inventory tracking, condition based maintenance, smart spaces, and military surveillance ([10], [9], [18], [5], and [12]).

The goal of a sensor network is to collect, process, and forward sensed data to other sensor nodes and/or base stations. Therefore, a routing service is essential to sensor network applications. Several non-traditional routing services have been proposed for sensor networks. They include Greedy Perimeter Stateless Routing (GPSR) [19], Geographical Energy Aware Routing (GEAR) [37], Trajectory Based Routing (TBF) [24], Directed-diffusion [17], Two-Tier Data-Dissemination (TTDD) [34], Content-Based Multicast (CBM) [39], Rumor-routing [3] and others. These routing services have distinct properties. First, they try to meet the resource-limited requirements of sensor networks [16]. Second, these routing services are different from traditional routing such as those in the Internet (OSPF [23], RIP [13] and BGP ([28]). One main difference is that data may be forwarded in sensor networks without explicitly addressing individual nodes. For example, GPSR takes locations instead of IP addresses as routing destinations. TBF uses a trajectory instead of a node-path as its routing unit. In addition, many routing services, such as Directed-diffusion, TTDD, and CBM, forward packets according to packet contents. Table 1 shows some other differences among routing services for sensor networks and traditional network routing.

However, each of the above routing services is designed to meet specific goals and therefore is not efficient for all applications. For example, Directed-diffusion has been shown to be more energy-efficient than TTDD when the number of sink nodes is large, while TTDD is better when the number of sink nodes is small [34]. In addition, different network conditions need different routing services. For example, GPSR is a good routing service when there is a scalable location service, but it cannot be used when sensor nodes have no location knowledge. In this case, if the sensor nodes are densely deployed, Rumor-routing can be a feasible routing choice.

There is a need to have a routing service for sensor networks that can adapt to different applications and different network conditions. To relieve management complexity, a routing service should have autonomic computing abilities that enable changes in an automatic and self-controlled way [11]. Currently, it is very difficult, if not impossible, to change a routing service in a large scale sensor network because the service is statically pre-configured into each node, which is often unattended.

This paper proposes a programmable routing framework that creates an adaptable routing service for sensor networks. In this framework, a routing service is divided into several programmable components. Based on this division, a universal routing service is developed that allows the introduction of different services through a set of tunable parameters and programmable components. We further propose a deployment service to deploy the universal routing service throughout a network. In order to automate the routing co[illegible]

PROCEEDINGS OF THE AUTONOMIC COMPUTING WORKSHOP FIFTH ANNUAL INTERNATIONAL WORKSHOP
ON ACTIVE MIDDLEWARE SERVICES (AMS'03) ISBN 0-7695-1983-0/01 $17.00 © 2003 IEEE

| | Supported Communication Mode | What trigger forwarding | State Type |
|---|---|---|---|
| **GPSR** | Unicast | Data packets | Neighbor location |
| **GEAR** | Unicast and Multicast | Data packets | Neighbor location |
| **TBF** | Unicast and Multicast | Data packets | Neighbor location |
| **Directed-diffusion** | Unicast, Multicast, and Many-to-one | Data packets | Neighbor info, neighbor interest, neighbor data copy |
| **TTDD** | Multicast | Data packets | Neighbor location, neighbor interest, neighbor data availability |
| **CBM** | Multicast | Control/Data packets | Neighbor info, neighbor interest, data from sources |
| **Rumor-routing** | Unicast | Data packets | Neighbor info, neighbor data availability, routing table |
| **Traditional routing** | Unicast or multicast | Data packets | Routing table |

**Table 1. Comparison of Routing Services in Sensor Networks and Traditional Networks**

ration/deployment in an energy-efficient way, this deployment service supports a three-level approach that differentiates the deployment of parameters and programmable modules, co-exists with a shared library to reduce the code size of programmable components, and provides a lightweight synchronization protocol to maintain configuration consistency throughout the network in a dynamic manner. With this deployment service, a self-configuring ability is realized for sensor routing services. With the changeable parameters and programmable components of the universal routing service, the self-optimizing as well as other autonomic abilities can be explored in an experimental sensor network conforming to our proposed framework.

This paper is organized as follows. Section 2 proposes a programmable routing architecture for sensor networks. Section 3 describes a universal routing service based on the architecture. In Section 4, we present how to deploy programmable routing services in large-scale sensor networks. Section 5 describes some related work. Finally, concluding remarks are given in Section 6.

## 2. A Programmable Routing Architecture for Sensor Networks

A routing service for sensor networks must be lightweight, due to limited available resources. This property prohibits a routing service in sensor networks from being constructed in a highly sophisticated way such as the ones in extensible routing architectures for traditional networks ([25], [8], [20], [7], and [36]). Sensor networks should favor a simple structure for a programmable routing service.

Figure 1 shows the sensor node architecture with the proposed programmable routing framework. The routing service is divided into three parts: a data-forwarding module, a state-collecting module, and state information. The data-forwarding and state-collecting modules are programmable. The state information is used by these two modules. In this architecture, we also propose a deployment service to load the programmable routing service, which will be presented with more detail in Section 4. This section focuses only on the programmable routing service.

Two types of packets can originate within or transit the routing service, namely: *data packets* and *control packets*. Data packets usually carry sensed data. Besides forwarding a data packet, the data-forwarding module can consume the data packet by sending it to an application-layer service, or construct a data packet by obtaining the data from the application-layer service. Control packets are constructed by the state-collecting module in a similar way and are typically used for status check (Table 1) and state information gathering to facilitate forwarding decisions. State information can also be collected by the data-forwarding module through observing data packets. The data-forwarding and state-collecting modules can communicate with each other not only through the state information, but also through direct signals. For example, when the state-collecting module receives a request, it can trigger the data-forwarding module to forward the requested data immediately through a signal (Table 1). Table 2 shows the data-forwarding and state-collecting functions of the existing routing services. It can be shown that each of them is covered by the programmable structure.

The above routing architecture provides a divide-and-conquer way to define a routing service. For example, the state information part should be scalable; the consumed space should not increase significantly with increase of network size and the number of source/sink pairs. As shown in Table 1, CBM keeps data from sources as state that is not scalable with the number of sources. Similarly, Rumor-routing maintains routing table as state information that is not scalable with the number of sinks unless a hierarchical routing structure is assumed. Solutions to these problems

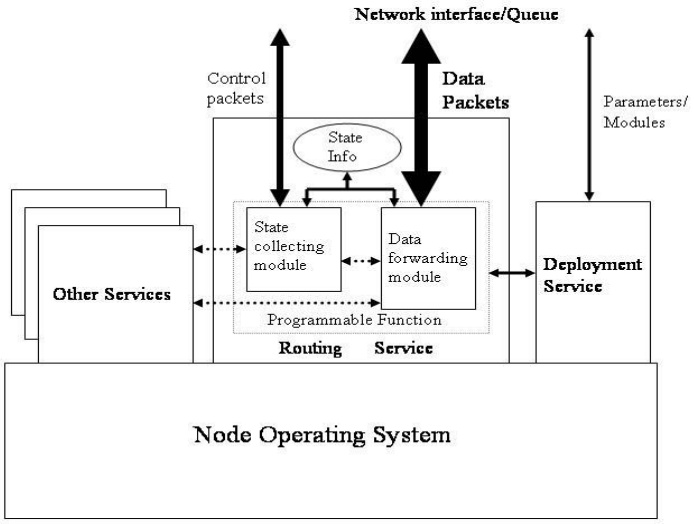


**Figure 1. Sensor Node Architecture with the Programmable Routing Framework**

will be discussed in the context of the proposed universal programmable routing service in Section 3.

The programmable architecture also clarifies the control packet overhead compared with overhead of data packets. A low-overhead routing service suggests that the total number of control packets be much less than the number of data packets. Besides, the average control packet size should be considerably smaller than the size of data packets. Consequently, we believe that one important research problem about routing services is the *visiting pattern* of control packets. For example, what frequency of the control packets are generated, what paths the control packets should take to go through the network to complete a state-building for a given amount of forwarded data, and which visiting pattern involves least overhead? The third column of Table 2 gives a brief view of the visiting patterns of existing routing services, and Figure 2 depicts four significant cases. One observation is that the query flooding in Directed-diffusion may cause much overhead. Another observation is that CBM sources flood data into certain areas to create state. This flooding must be done restrictively since flooding data involves significant overhead. Section 3 will discuss visiting patterns in more detail.

## 3. A Universal Programmable Routing Service for Sensor Networks

Based on the routing architecture in Section 2, a universal programmable routing service is proposed to cover all existing routing services and to introduce new services for sensor networks. The components of this universal service, a tunable state information part, a programmable state-collecting module, and a programmable data-forwarding module, are as follows.

The state information is a list of neighbor entries, each of which consists of four parts:

- neighbor description (id, location, direction, distance, energy reading, etc.)
- neighbor interest (type, rate, duration, etc.)
- neighbor data availability (type, duration, etc.)
- neighbor's latest data copy (data, timestamp, etc.)

The above state involves only local information and thus is scalable. This property solves the scalability problems with state of Rumor-routing and CBM. For Rumor-routing, the routing table is replaced by neighbor interest so that the state does not increase with the number of sinks. For CBM, the data copies from sources are replaced by data copies from neighbors. This change is reasonable since the sinks only are concerned with what the data are and not who sends them. In this way, CBM and Rumor-routing become scalable with our universal service.

Different packets are used to collect each part of the state information. The following list shows the type of packet for each state component:

- neighbor description - hello/announcement/query packets
- neighbor interest - query packets
- neighbor data availability - announcement/data packets
- neighbor latest data copy - data packets

The *visiting pattern* of each packet type is an important parameter to control the overhead of the state collection. For hello packets, the visiting pattern is straightforward and has low overhead. For data packets, the visiting pattern generally follows the route indicated by the neighbor interest state. But when the data packets are solely used to build state, as the case in CBM, the visiting pattern of these data packets needs to be configured. For these state-collecting data packets, as well as announcement and query packets, visiting pa[illegible]

| | Data Forwarding | Routing State Collection |
|---|---|---|
| **GPSR** | Closest or perimeter neighbor to destination | Hello packets |
| **GEAR** | Closest to destination(s) or multiple neighbor(s) | Hello packets |
| **TBF** | Neighbor(s) along trajectory | Hello packets |
| **Directed-diffusion** | Interested neighbor(s) with requested rate(s) | Query flooding with reinforcement |
| **TTDD** | Interested grid cross-point neighbor(s) | Data announcement along grids + query local flooding |
| **CBM** | Interested neighbor(s) | Data restrictive flooding + query restrictive flooding |
| **Rumor-routing** | Next hop in routing table | Data announcement/query with a random path |

**Table 2. Data Forwarding and State Collecting Functions in Routing Services for Sensor Networks**

can be specified as flooding, restricted flooding, single path, or none. Among the four choices, both the restricted flooding and the single path options have numerous possibilities. For example, when location information is available, these two options can be specified via certain trajectories such as those in [24]; when location information is not available, some probabilistic method can be used [3].

The final part of the universal routing service is the data-forwarding module. The basic function of this module is to check data packets against the state information and select among neighbor(s) to forward the data. It also can include application-specific in-network processing such as aggregation [22] of the data. Its tunable parameter is a set of *selection criteria* that includes: whether to check packet header or content, how to choose forwarding neighbors, and some QoS specification (say, data rates requested by neighbors).

Note that the state-collecting module, the data-forwarding module, and the entire function (Figure 1) of the universal service are programmable. Once the interfaces among them and other services are defined, all of these three are changeable, which brings a significant flexibility to adapt to applications and introduce new services. Besides, changing parameters in the proposed universal service allows changing among available services and introducing new services. For example, by changing the *visiting pattern* and/or *selection criteria* parameter(s), we can obtain significantly different services (Table 2). Identifying these two parameters in the universal service also facilitates studies about autonomic computing in routing services. Suppose we want to obtain new routing services that can automatically change selections among neighbors based on certain QoS requirements, and/or adapt packet-visiting paths based on learned routing overhead. One way to achieve this is to leverage the deployment method in Section 4 to try out different parameter values. Then some heuristic strategies learned from these experiments can be added to the routing service to realize good performance.

# 4. Deployment of Programmable Routing Services

This section first gives an overview of the proposed deployment service shown in Figure 1. It then discusses approaches to reduce communication overhead for routing service deployment and presents how to maintain service consistency throughout a sensor network in a dynamic way.

## 4.1. Overview

A few papers have described deployment in programmable sensor networks ([21], [2], and [1]). The general approach proposed in these papers is to design some specific instruction that enables a service to replicate itself onto other nodes. To support these instructions, each node contains a runtime environment for the service. In this way, a service can be loaded on a subset of nodes to carry out a distributed task. This approach may be useful to deploy certain transient user-level services, but may not be feasible to deploy a routing service in a sensor network. First, services deployed by this approach are running (interpreted) on top of runtime environment, and suffer computation overhead ([21] and [1]). This computation overhead may not significantly affect the performance of a transient user-level application, but the performance of a routing service would be substantially degraded since routing services are persistent and computation-intensive. Second, a routing service needs to be deployed to every node in the network instead of just a subset of nodes. The specific replication instructions are not necessary to deploy routing services.

To deploy a programmable routing service through a sensor network, we can use a flooding protocol to disseminate the service from any node to all other nodes. One example of such a broadcast protocol is provided by SPIN (Sensor Protocols for Information via Negotiation) [15]. However, reliability would need to be added because the deployment may involve transferring binary code, which must be reliable. In

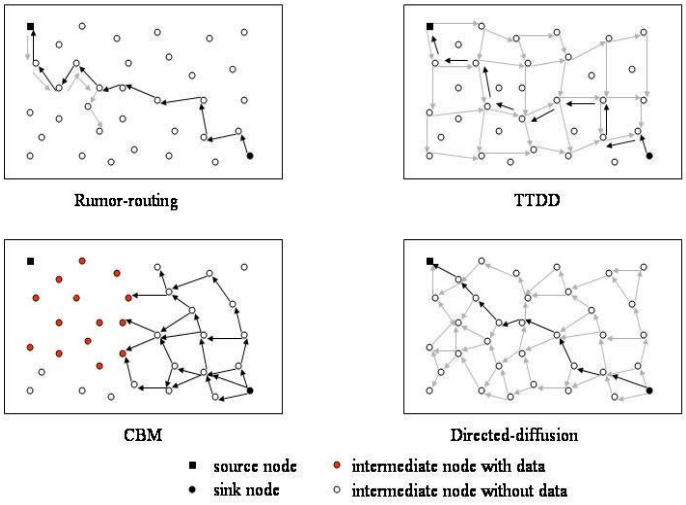


**Figure 2. Visiting Patterns of Control Packets for Four Routing Services**

a very large sensor network, the deployment can take advantage of a hierarchical structure that is provided either statically or dynamically. For example, we can use base stations scattered around the network, each in charge of a sub-area of a sensor field. If there are no base stations available, one natural replacement is beacon nodes used in self-localization systems [4] or header nodes in any other self-clustering system [10]. These nodes are dynamically maintained so that each sensor node is close to at least one such node. Deployment of routing services can be accomplished by deploying these high-level nodes first.

In our routing framework, each sensor node contains a deployment service as shown in Figure 1. This deployment service receives deployment packets that contain parameters or modules of the programmable routing services and deploy services according to packet contents. This service performs three levels of deployment: (1) the deployment service only changes parameters to the state-collecting and/or data-forwarding modules; (2) either of the two modules is replaced; (3) the entire routing service is changed. The first level of deployment obviously has least bandwidth requirement and thus can be done relatively frequently. The third level of deployment has most overhead and should be done only occasionally, say, when initializing sensor networks. The second level of deployment is a middle case and can be used when verifying some improvement on existing modules. Obviously, this deployment service, plus the proposed universal programmable routing service, provides a unique framework to conduct a systematic comparative experimental study on routing services for sensor networks.

Note that this deployment service allows different routing services to reside in different parts of sensor networks. One possible use of this property is for heterogeneous sensor networks. For example, GPSR service and Rumor-routing service, mentioned in Section 1, can be deployed in heterogeneous parts of a sensor network. Generally, a different routing service can be deployed at different condition. This flexibility allows us to study the changing behaviors of routing services in order to obtain an adaptive routing service that performs well under different conditions.

### 4.2. Reducing Communication Overhead for Deployment

A routing service may be complex and require large code. Transferring the routing code can be expensive in sensor networks where energy is a very scarce resource. Although the module deployment (corresponding to the second-level and the third-level deployment methods) for routing services does not happen frequently, it is still beneficial to deploy modules in an energy-efficient way.

To reduce service code size, [1] proposes a separate running environment that provides high-level building blocks for services. Services running on top of the environment then are expected to have small code size. This approach is not desirable for routing services due to the entailed computation inefficiency. A deployment approach for routing services should be both energy-efficient and computation-efficient.

The approach we take is to move a part of routing service code into sensor nodes before deploying routing service modules. This part of code contains common routing service operations and is designed as a *shared library* [32]. Routing modules such as the state-collecting and the data-forwarding modules are written by calling the shared library. Since the library code will not be loaded until the routing modules start to run, routing modules written with the library have smaller code sizes than those without the library. On the other hand, library code is automatically loaded by node OS when routing modules start. The whole routing service still runs directly on top of node OS. Thus the computation efficiency is kept.

Figure 3 shows a sample node architecture extended from Figure 1 for the purpose of reducing communication overhead. For simplicity, Figure 3 does not show interactions

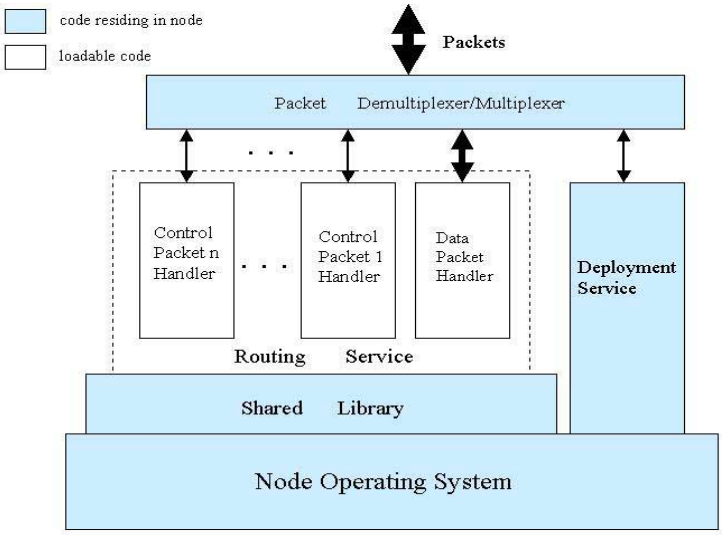


**Figure 3. A Sample Node Architecture with Shared Library**

among the routing service and other services. The gray boxes in Figure 3 are code that is pre-configured before deploying a routing service, while the blank boxes are programmable components for the routing service. In Figure 3, a routing service is decomposed with a finer granularity than Figure 1. The decomposition is done according to packet type; the state-collecting module is decomposed into several handlers that process different control packets. Note this decomposition does not significantly increase the complexity of a routing structure since there are generally only a few packet types for a certain routing service (for example, there are only six packet types for Directed-diffusion [29]). This decomposition further reduces the code size for each deployment unit and increases service flexibility. Figure 3 also shows a packet demultiplexer/multiplexer whose parameters (say, packet types) are changeable via the deployment service. This packet demultiplexer/multiplexer directly runs on top of node OS and can be integrated with the deployment service (not shown).

Note that routing modules do not necessarily have to be built with the shared library. They can be built from scratch when the library does not provide required operations. Of course, this may increase the deployment cost. Thus the library code should also be changeable, with a frequency higher than the OS but lower than a routing service.

### 4.3. Maintaining Deployment Consistency

The deployment service described so far assumes that all nodes in a network can be reached at one time, but this is generally not the case for sensor networks. Sensor networks are prone to sensor node failures due to running out of energy, and communication failures due to lossy channel or obstacles. Besides, it is normal for a sensor node to periodically or dynamically sleep for some time because of using energy-saving mechanisms such as those in [31], [33], and [35]. The consequence of these dynamics is inconsistency among nodes for deployed services.

We propose a synchronization protocol that enables a sensor node to make itself consistent with its neighbors in an energy-efficient way. Each node runs this protocol after waking up from sleeping or after a period. Since a routing service is updated less frequently than user-level applications, this period can be relatively long, say, multiple times of a hello packet interval.

Each node maintains a version number for each deployed component (a parameter or a module). To start this protocol, a node (say, $A$) first broadcasts an *initial request* among its neighbors for each component it wants to synchronize. The initial request includes the node's version number for the component. Upon receiving the request, each neighbor $N_i$ that has a version number greater than the requesting node starts out a timer. The timeout value can be determined by the following example formula:

$$Timeout(N_i) = \alpha/|V(N_i) - V(A)| + \beta/E(N_i) + R \quad (1)$$

where $\alpha$ and $\beta$ are tunable weights, $V(N_i)$ is the version number of $N_i$, $V(A)$ is the version number of $A$, $E(N_i)$ is the remaining energy of $N_i$, and R is a random variable to break ties between nodes that have the same version number and the same remaining energy. This timeout mechanism ensures that the neighbor with the highest version number and remaining energy first times out. After the timeout, the neighbor sends an *initial reply* to the requesting node, suppressing other overhearing nodes. The requesting node also starts a timer immediately after sending the initial request to wait for all possible initial replies from its neighbors. After collecting all initial replies from neighbors, the requesting node chooses the node with highest version number to send a *formal request*, which in turn sends back a *formal reply* to complete the synchronization. If the requesting node does not receive any initial reply before the timeout, it assumes it has updated routing service.

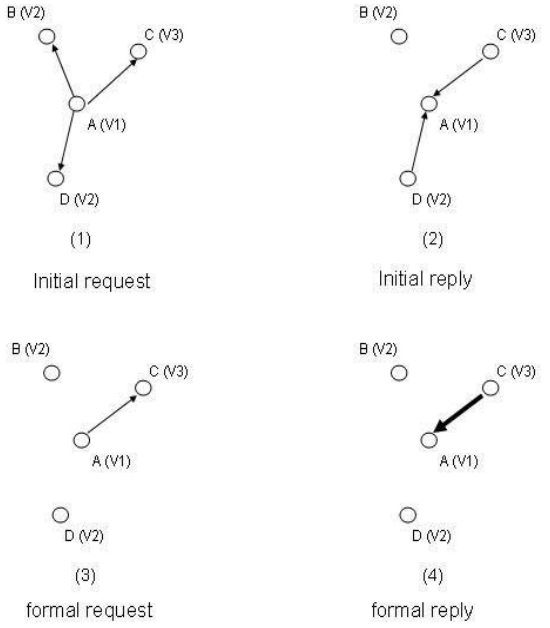


**Figure 4. Deployment Synchronization from Neighbors**

Figure 4 shows an example for the synchronization protocol. Node $C$ has the highest version number and will time out first. Since Node $B$ is in the range of Node $C$, its reply is suppressed. But Node $D$ is out of range of Node $C$, thus it also sends an initial reply to Node $A$. After obtaining all replies, Node $A$ chooses Node $D$ to finish the synchronization.

Note that the initial request can be piggybacked on hello packets. Thus this protocol causes no additional communication cost when there is consistency. In case of inconsistency, there is only one neighbor taking part in the updating process. The proposed protocol ensures that each node eventually obtains the latest version of the routing service with low communication overhead as long as the network is not always disconnected. This synchronization protocol together with the flooding protocol described in Section 4.1 complete an automatic configuration for a routing service in a sensor network.

## 5. Related Work

Scout [25], Router-Plugins [8], Click [20] and Internet Exchangeable Architecture [7] study extensible router architectures [36] to meet the needs of sophisticated services in the Internet. These routers follow powerful and heavyweight models and are not applicable to sensor networks where resources are very constrained. Besides, it is substantially difficult to deploy extensible routers on the Internet due to their significant changes to traditional routers that do not have an open architecture. On the contrary, our proposed routing framework follows a lightweight structure, and its deployment is relatively easy since sensor networks explicitly favor an open architecture.

Filter-based architecture in [14] specifies a software structure that contains a list of filter handlers, each of which is executed when its attributes match with incoming packets. This architecture provides a flexible way to add application-specific code into sensor nodes. By following its API [30], users of a sensor network can write their own in-network processing modules and add them to the network in the form of filters. One good example of such application-specific filters is the information-driven tracking filter ([38] and [6]) that uses tracking information to conserve energy. Our proposed framework is related to this architecture. For example, the universal routing service can be considered a routing filter in the filter-based architecture. But our framework studies the routing filter in a finer granularity. A routing service in our framework is decomposed into several programmable components, and is thus more general and flexible. Note that the core part of the filter-based architecture is based on Directed-diffusion and cannot be changed by users. Thus it is impossible to implement other significantly different routing services such as TTDD by means of filters. Another benefit of our framework is that the decomposition allows for an deployment method with communication overhead substantially lower than loading a single big filter. The current implementation of the filter-based architecture [29] does not support the automatic deployment of filters, all of which have to be statically pre-configured into nodes.

In [21], a virtual machine with a small instruction set is built for each sensor node. Services constructed via this instruction set are interpreted by this virtual machine. Each instruction in the set abstracts some high-level function carried out by TinyOS [16]. The goal is thus to provide concise code for sensor services that can be loaded in a memory-and-energy-saving manner. This virtual machine could be a potential environment to implement routing services. However there are two limitations associated with the current implementation of this virtual machine. First, the expressiveness of its instruction set is limited. For example, [1] has shown that it is difficult to write an aggregation service with this instruction set due to its limited stack size. With this ultra-compact instruction set, it may be difficult to write significant routing services for sensor networks. Second, this virtual machine entails significant computation overhead to services built on top of it. This virtual machine may not be used as an developing platform for the proposed routing framework before the above two limitations are overcome.

Sensorware ([2] and [1]) is another runtime environment but provides a higher-level service abstraction than the virtual machine proposed in [21]. Services then can have smaller code size, written using a script language that can be understood by the sensorware built in each node. The sensorware targets to support a specific kind of application called localized distributed application such as a collaborative signal processing task. These applications generally only involve a subset of sensor nodes and are transient. Further

more, due to reasons similar to the virtual machine in [21], using scripts to describe a routing service is not computationally efficient. We believe that loading decomposed code with a shared-library support is a both energy and computationally efficient way to deploy a routing service in sensor networks. Note that the sensorware can co-exist with our proposed framework. It is necessary to separate the deployment of persistent system-level services and transient user-level services since these two kind of services have different requirements about computation and energy efficiency. Our deployment service can be integrated with deployment of other system services and user-level services (such as sensorware).

## 6. Conclusions

We have described a universal routing service based on a programmable architecture for sensor networks. We have identified two parameters (visiting pattern and selection criteria) for the universal service whose changes facilitate the optimization of a routing service. With its programmable components, the universal service can freely change its behavior. A deployment service has been proposed to deploy the universal routing service throughout a network. This deployment service has taken four approaches to automatically configure a network in an energy-efficient way. First, a reliable broadcast protocol has been included to disseminate a routing service. Second, a three-level deployment method has been presented to differentiate deployment between parameters and programmable modules. Third, to reduce the communication overhead for module deployment, we have proposed a shared-library mechanism that does not incur additional computation overhead. Finally, a lightweight synchronization protocol has been described to maintain service consistency throughout the network in a dynamic way.

With the presentation of our framework, we have shown that the configuration complexity for routing services in sensor networks can be significantly relieved by an automatic deployment service. The self-optimizing as well as other autonomic abilities of a routing service can be explored by experimenting with the tunable parameters and programmable components of the universal service with a sensor network conforming to our proposed framework.

## References


[1] A. Boulis, C.-C. Han, and M. B. Srivastava. Design and implementation of a framework for efficient and programmable sensor networks. *MobiSys*, May 2003.

[2] A. Boulis and M. B. Srivastava. A framework for efficient and programmable sensor networks. *OPENARCH*, July 2002.

[3] D. Braginsky and D. Estrin. Rumor routing algorithm for sensor networks. *WSNA*, September 2002.

[4] N. Bulusu, J. Heidemann, and D. Estrin. Self-configuring localization systems: Design and experiment evaluation. *Submitted to ACM TECS Special Issue on Networked Embedded Computing*, August 2002.

[5] J. Cerpa, D. Estrin, and et al. Habitat monitoring: application driver for wireless communications technology. *ACM SIGCOMM Workshop on Data Communications in Latin America and the Caribbean*, April 2001.

[6] M. Chu, H. Haussecker, and et al. Scalable information-driven sensor querying and routing for ad hoc heterogeneous sensor networks. *Int'l J. High Performance Computing Applications*, 16(3), Fall 2002.

[7] I. Corporation. Ixp1200 network processor datasheet. September 2000.

[8] D. Decasper, Z. Dittia, G. Parulkar, and B. Plattner. Router plugins: A software architecture for next generation routers. *IEEE/ACM Transactions on Networking*, 8(1):2–15, Februray 2000.

[9] D. Estrin, L. Girod, and et al. Instrumenting the world with wireless sensor networks. *ICASSP*, May 2001.

[10] D. Estrin, R. Govindan, and et al. Next century challenges: scalable coordination in sensor networks. *Mobicom*, August 1999.

[11] A. Ganek and et al. Autonomic computing: Ibm's perspective on the state of information technology. *http://www.research.ibm.com/autonomic/manifesto/*, October 2001.

[12] L. Girod, D. Estrin, and et al. Robust range estimation using acoustic and multimodal sensing. *IEEE Conference on Intelligent Robots and Systems*, 2001.

[13] C. Hedrick. Rfc 1058 - routing information protocol. *Request for Comments*, June 1988.

[14] J. Heidemann, F. Silva, and et al. Diffusion filters as a flexible architecture for event notification in wireless sensor networks. *USC/ISI Tech. Report*, April 2002.

[15] W. Heinzelman, J. Kulik, and H. Balakrishnan. Adaptive protocols for information dissemination in wireless sensor networks. *Mobicom*, August 1999.

[16] J. Hill and et al. System architecture directions for networked sensors. *ASPLOS*, November 2000.

[17] C. Intanagonwiwat, R. Govindan, and D. Estrin. Directed diffusion: A scalabel and robust communication paradigm for sensor networks. *Mobicom*, 2000.

[18] J. M. Kahn and et al. Mobile networking for smart dust. *Mobicom*, August 1999.

[19] B. Karp and H. T. Kung. Gpsr: Greedy perimeter stateless routing for wireless networks. *Mobicom*, 2000.

[20] E. Kohler and et al. The click modular router. *ACM Transactions on Computer Systems*, 18(3):263–297, August 2000.

[21] P. Levis and D. Culler. Maté: A tiny virtual machine for sensor networks. *ASPLOS X*, October 2002.

[22] S. R. Madden, R. Szewczyk, M. J. Franklin, and D. Culler. Supporting aggregate queries over ad-hoc wireless sensor networks. *WMCSA*, June 2002.

[23] J. Moy. Rfc 2328 - ospf version 2. *Request for Comments*, April 1998.

[24] B. Nath and D. Niculescu. Routing on a curve. *Hotnets I*, October 2002.

[25] L. Peterson and et al. An os interface for active routers. *IEEE J-SAC*, 19(3):473–487, March 2001.

[26] K. Pister and et al. Smart dust: wireless networks of milimeter-scale sensor nodes. *Highlight Article in 1999 Electronics Research Laboratory Research Summary*, 1999.

[27] G. Pottie and et al. Wireless sensor networks. *Communications of the ACM*, 2000.

[28] Y. Rekhter and T. Li. Rfc 1771 - a border gateway protocol (bgp-4). *Request for Comments*, March 1995.

[29] F. Silva and et al. Diffusion 3.1.3 code. *http://www.isi.edu/scadds/software/*, July 2002.
[30] F. Silva, J. Heidemann, and et al. Network routing application programmer's interface (api) and walk through 9.0.1. *http://www.isi.edu/scadds/papers/*, December 2002.
[31] S. Singh and C. S. Raghavendra. Pamas - power aware multi-access protocol with signalling for ad hoc networks. *ACM Computer Communication Review*, July 1998.
[32] D. A. Wheeler. Program library howto. *http://www.tldp.org/HOWTO/Program-Library-HOWTO/*, December 2002.
[33] Y. Xu, J. Heidemann, and D. Estrin. Geography-informed energy conservation for ad hoc routing. *Mobicom*, 2001.
[34] F. Ye, H. Luo, J. Cheng, S. Lu, and L. Zhang. A two-tier data dissemination model for large-scale wireless sensor networks. *Mobicom*, September 2002.
[35] W. Ye, J. Heidemann, and D. Estrin. An energy-efficient mac protocol for wireless sensor networks. *Infocom*, June 2002.
[36] L. P. Yitzchak Gottlieb. A comparative study of extensible routers. *IEEE OPENARCH*, June 2002.
[37] Y. Yu, R. Govindan, and D. Estrin. Geographical and energy aware routing: a recursive data dissmination protocol for wireless sensor networks. *UCLA CS Tech. Report*, 2001.
[38] F. Zhao, J. Shin, and et al. Information-driven dynamic sensor collaboration for tracking applications. *IEEE Singal Processing Magazine*, March 2002.
[39] H. Zhou and S. Singh. Content-based multicast for mobile ad hoc networks. *Mobihoc*, August 2000.