Reprogramming Wireless Sensor Networks: Challenges and Approaches

Qiang Wang, Yaoyao Zhu, and Liang Cheng, Lehigh University

Abstract

Wireless sensor networks need an efficient and reliable reprogramming service to facilitate management and maintenance tasks. In this article we first outline a framework to examine different functions in reprogramming, followed by an analysis of reprogramming challenges. We then provide a comprehensive survey of the state-of-the-art reprogramming systems, and discuss different approaches to address these challenges. Finally we explore performance, protocol behavior, and the impact of several design factors.

A typical wireless sensor network (WSN) consists of a large number of small-sized battery-powered sensor nodes that integrate sensing, computing, and communication capabilities. WSN applications include geophysical/structural/habitat monitoring, security surveillance, disaster area or battlefield information collection, and pervasive computing. In most applications sensor networks are deployed once and intended to operate unattended for a long period of time. Management and maintenance tasks of WSNs are challenging. Enabling sensor networks to be reprogrammable is a way to address such challenges.

Code dissemination and code acquisition are two basic schemes to reprogram sensor networks. Code dissemination is usually used by system administrators for updating programs on sensor nodes, fixing bugs, changing network functionality, tuning module parameters, and replacing program modules. Traditional ways of manually reprogramming sensors are costly, labor intensive or even impossible since each node has to be collected from the field and physically attached to a computer to "burn" new codes. In contrast to code dissemination, code acquisition is initiated from individual sensors to fetch and install program modules from the network dynamically and on demand. It enables sensor nodes to self-reprogram so that they can adapt to changing tasks and evolving environments. Due to capacity constraints of sensor nodes, a monolithic program with too many functions cannot be fitted into the memory. In addition, applications may need extra modules to handle unforeseen events.

Reprogramming usually is implemented as an application-independent service on top of a sensor operating system. This article reviews the reprogramming framework, design challenges, existing systems and approaches, evaluation metrics, and system behaviors.

WSN Reprogramming Framework

There are several major functions related to WSN reprogramming, as outlined in a framework in Fig. 1.

Version control: A version control database manages various program versions, the relations between update patches, information about the current running programs on sensor nodes, and so on.

Scope selection: Scope selection allows administrators to select any particular nodes in the network for reprogramming. The scope of reprogramming can also be the whole network.

Encoding/decoding: In simplest form, a reprogramming system reads the new program image, encodes it into data packets, and sends packets out through radio. A receiver decodes these packets and rebuilds the program image. Improved encoding/decoding can be used to reduce the size of updates and traffic loads in a WSN. It can also add security to data transmission.

Code dissemination: This protocol transmits updates from source nodes to targeted receivers. Code dissemination and scope selection can work together to reprogram a partial network.

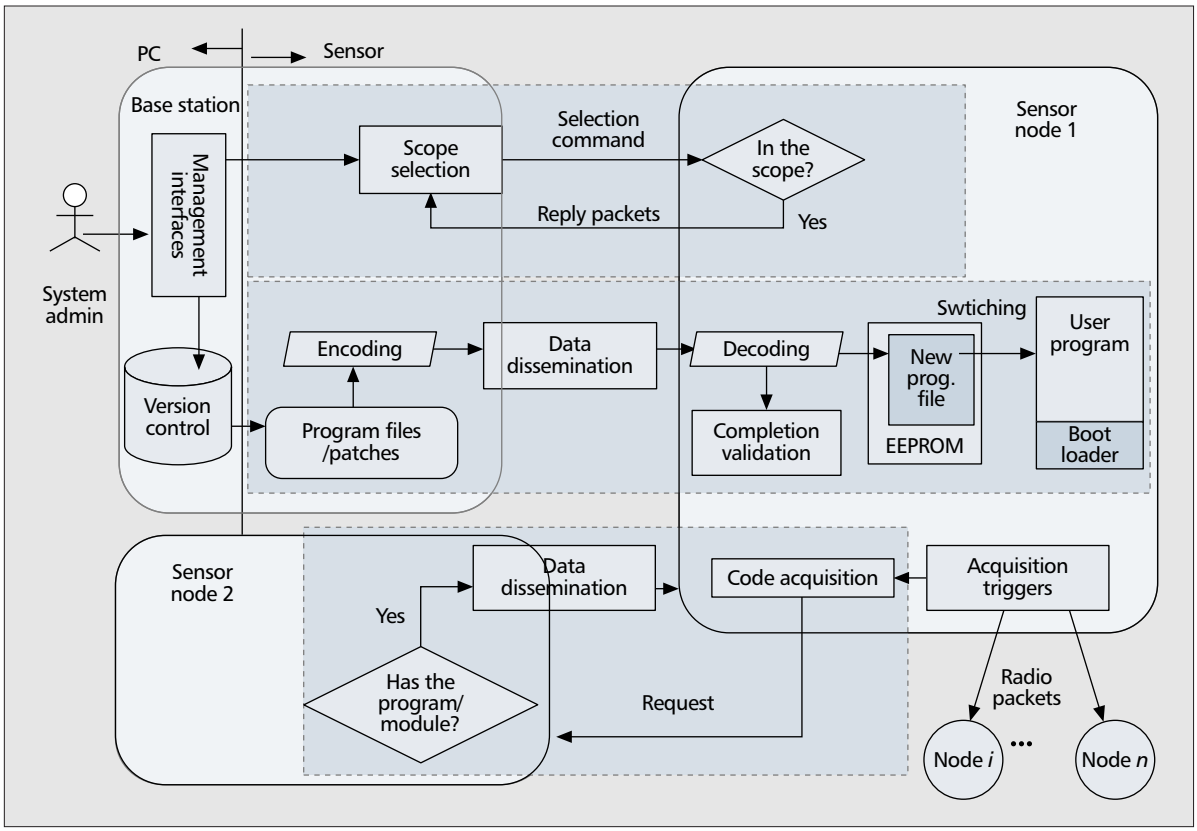
Completion validation: The new program should be received in its entirety without errors before running. Completion validation ensures the continuous functioning of WSNs.

Code acquisition: Code acquisition is initiated from sensor nodes. Trigger conditions could be set in sensor nodes or computed at runtime. If a condition is satisfied, a sensor node will send code acquisition requests to find source node(s) that have the desired program, module, or patch. After that, route(s) will be built to send the codes from the source node(s) to the requesting node.

Switching: After being received over the air, the new program codes are usually stored in an external flash memory (EEPROM) because the data memory is not large enough. A sensor node will switch to the new program, typically done through a reboot. A boot loader, which resides in a specific section of the microcontroller's program memory, copies the received program code from EEPROM to the user application section of the program memory. Then the new program starts to execute. However, before switching to the new program, the node usually needs to preserve some data and states (e.g., node ID, group ID, neighbor lists, routing tables, and calibration parameters), and restores them into the new program. The address dependency of these data is an important factor to design the store and restore mechanisms; for example, a routing table from the old program must be restored to its appropriate address space in the new program.

Management interface: The interface is installed at a base station (a PC with an attached sensor node) for administrators

0890-8044/06/$20.00 © 2006 IEEE



■ Figure 1. *A WSN reprogramming framework.*

to perform reprogramming tasks (initiate/monitor/cancel reprogramming, setting parameters for scope selection, etc.).

WSN Reprogramming Challenges

In order to develop a practical and efficient reprogramming system, many challenges have to be addressed, largely due to the characteristics of WSNs and wireless communications. We particularly examine TinyOS and Mica Mote in this section, as they are now popular development environments for WSN applications.

Characteristics of Wireless Sensor Networks

Unlike reprogramming on other platforms (e.g., PCs), a reprogramming service for WSNs must run on sensor nodes with very limited resources.

First, the time and space complexity of algorithms in reprogramming should be well fitted to the capacity profile of a sensor node. Sensor nodes are generally small in size with limited hardware capacities. For example, Mica2's ATmega128L microcontroller has 8 MIPS throughput at 8 MHz. Mica2 has merely 4 kbytes data memory, 128 kbytes program memory, and 512 kbytes external flash memory (EEPROM) [1].

Second, reprogramming should be energy-efficient. Sensor nodes are usually battery powered and can hold/gain limited amounts of energy. Among computing, communication, and sensing functions, communication consumes a large portion of the energy, as shown in Table 1 [1]. Writing operation to the EEPROM is also expensive. Idle listening also consumes energy and should be avoided as much as possible.

Third, reprogramming requires the program code to be delivered in its entirety. However, wireless communication is unreliable due to possible signal collisions, interferences, and packet contentions. Furthermore, network topologies may change due to node failures, node mobility, and so on, which makes reliable protocol designs more challenging.

Fourth, scalability is crucial for large-scale sensor network deployment. Scalability has two requirements for a widely applicable reprogramming service:

- Scale for number of nodes, from tens up to hundreds or even thousands of nodes
- Scale for varying node density, from sparse to dense networks

Fifth, there are several programming support limitations in current TinyOS:

- A compiled program is monolithic. For efficient execution on Mote devices, application modules and TinyOS kernels are statically compiled, globally optimized, and intertwined into a single executable image. There is no individual module that can easily be separated and then reprogrammed independently.
- There is no reliable dynamic memory allocation. Reprogramming services and other applications have to share the scarce 4 kbytes data memory and 128 kbytes program memory.

Operations	Power consumption (nAh)
Read a data block from EEPROM	1.261
Write a data block to EEPROM	85.449
Send one packet	20.000
Receive one packet	8.000
Idle listen for 1 ms	1.250

■ Table 1. *Typical power consumption of Mica2 Motes.*

Name	Encoding/decoding	MAC	Hop	Scope	Hierarchy	Pipelining
XNP [1]	Complete program	CSMA	Single-hop	Whole network	No	No
Reijers approach [4]	Platform-dependent patch	CSMA	Single-hop	Whole network	No	No
Incremental [5]	Platform-independent patch	CSMA	Single-hop	Whole network	No	No
Trickle [6]	Maté script	CSMA	Multihop	Whole network	No	No
MOAP [7]	Complete program	CSMA	Multihop	Whole network	No	No
Deluge [8]	Complete program	CSMA	Multihop	Whole network	No	Yes
MNP [9]	Complete program	CSMA	Multihop	Whole network	No	Yes
Sprinkler [10]	Complete program	TDMA	Multihop	Whole network	Yes	No
Firecracker [11]	Complete program	CSMA	Multihop	Whole network	Yes	No
Aqueduct [12]	Complete program	CSMA	Multihop	Selected nodes	No	Yes
TinyCubus [13]	Modular update	CSMA	Multihop	Selected nodes	No	No

■ Table 2. *Available reprogramming systems for WSNs.*

- File systems and memory management units are currently unavailable, making safe memory and EEPROM access a nontrivial task.
- Network supports in current TinyOS release are limited: the MAC layer uses simple carrier sense multiple access (CSMA), and there is no reliable transport layer. The default maximum packet size is relatively small (e.g., less than 30 bytes as payload). Such small packet size profoundly impacts dissemination protocol designs.
- TinyOS supports a limited concurrency model. There is no multiprocess or multithread concept. Reprogramming implementation and debugging are also particularly difficult.

Broadcast Storm and Hidden Terminal Problems

A straightforward protocol of code disseminating in WSNs is classic flooding: a base node broadcasts new codes to its neighbors. Upon receiving the data, each node stores and then rebroadcasts to its neighbors. However, in large-scale WSNs with high density and limited energy, classic flooding is particularly costly and results in serious redundancy, contention, and collision. It is called the "broadcast storm" problem [2], and is mainly caused by two deficiencies:

- Data redundancy (implosion [3]): A sender may send out unnecessary (e.g., already received) data to its neighbors. To reduce data redundancy, a sender should be aware of what data has already been received by its receivers.
- Sender redundancy: Some senders are redundant to cover a desired area. These nodes cannot offer additional coverage (i.e., nodes that have not been covered by other broadcasts).

The hidden terminal is another issue in wireless communications. If two nodes are out of the transmission range of each other (thus "hidden" to each other), when they send packets at the same time, it may result in packet collisions at any node located within the intersection area of these senders. The hidden terminal problem degrades the performance of CSMA MAC substantially because carrier sensing cannot prevent collisions.

Reprogramming System Designs

In this section we first overview and classify the state-of-the-art reprogramming systems, then discuss various approaches that have been used to address the above-mentioned challenges. Finally, we specify some features of current systems that have not been widely explored, such as scope selection and code acquisition.

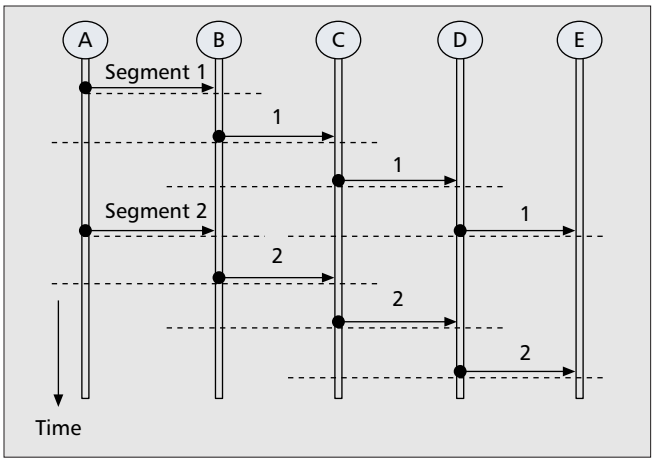
Overview of Reprogramming Systems

Several reprogramming systems have been designed and studied in the past few years, as summarized in Table 2. Except Sprinkler, all of them are designed for Berkeley TinyOS/Mote platform. These systems can be classified according to several criteria.

Single-hop vs. multihop: The earliest reprogramming systems, such as XNP [1], disseminate codes only within the radio communication range of a base station. To be more practical, reprogramming systems are now targeted at multihop scenarios. Multihop code dissemination protocols are epidemic in nature, and need to address important issues such as efficiency and scalability.

Encoding: Most reprogramming systems disseminate the compiled program image across the network. The overhead is usually large in cases when only minor changes occur between the new and old versions. Incremental update approaches compare the difference between the old and new programs, and only transmits the "delta patch." Incremental Network Programming [5] uses a differing algorithm (Rsync) optimized for sensors and assumes no prior knowledge of the program code structure (hardware independent). Reijers *et al.* use semantics of a particular microcontroller instruction set to repair address shifting, which makes their approach hardware-dependent [4]. Trickle [6] transmits Maté virtual machine scripts instead of native nesC compiled codes since Maté scripts are much smaller and simpler to write. However, Maté scripts are limited to the function of the virtual machine, and are not as flexible as nesC programming. TinyCubus [13] and SOS [14] compile an application into several module files, therefore only affected modules are transmitted in reprogramming. Appropriate lightweight data compression algorithms could also be applied to reduce the data size.

MAC: Since CSMA MAC is in the TinyOS release, most reprogramming systems use CSMA. Sprinkler [10] uses time-division multiple access (TDMA) MAC to reduce contention and achieve higher throughput. However, TDMA demands



■ Figure 2. *Pipelining two segments in a four-hop network.*

careful scheduling of time slots, and its implementation on a sensor platform is much more complex than CSMA. TDMA is not currently available for most sensor platforms.

Hierarchy: Sprinkler [10] and Firecracker [11] first send codes to nodes in the upper layer of the node hierarchy (i.e., super nodes). Then super nodes reprogram other nodes in their local areas. Super nodes can be cluster head nodes, or a set of connected dominating nodes that are sufficient to cover the whole network, as in Sprinkler. Super nodes in Firecracker are nodes in each corner, or randomly selected. MNP, Deluge, and others all start reprogramming from a single base station in the network and assume no hierarchy.

Scope: Most reprogramming systems have no scope selection function and only disseminate one program to the whole network. Aqueduct [12] and TinyCubus [13] support scope selection for targeted nodes.

Pipelining: Deluge and MNP support pipelining to accelerate reprogramming in multihop networks (discussed in the next section), while MOAP uses a sliding window concept.

Segmentation and Pipelining

Several reprogramming protocols take advantage of pipelining to allow parallel transfers of data in networks, such as Deluge and MNP. Pipelining is done through segmentation: a program is divided into several segments (called pages in Deluge), each of which contains a fixed number of packets. Instead of completely receiving a whole program before forwarding it, a node becomes a source node after it receives only one complete segment. For transmission of a large program, pipelining could increase overall throughput significantly. Other than pipelining, there is another major benefit of segmentation: without segmentation, a large program has thousands of packets. As a consequence, each node needs a large number of states to record the packet information (received or not).

Several issues need to be considered for achieving the full benefit of pipelining. One is to ensure that parallel transfers should not interfere with each other. In Fig. 2, corresponding to a simple channel model, a dashed line represents the interference range, and a solid arrow represents the reliable communication range. Due to the hidden terminal problem, the simultaneous data transfer from A to B will collide with the one from C to D. The parallel transfers should take place with at least three-hop spacing. In a network of n hops, without pipelining, transferring m segments needs a total time of $O(n * m)$. With pipelining, the total time is reduced to $O(n + 3 * (m–1))$.

One potential issue with implementing pipelining is that the entire network is powered on to support the pipelining, as done by Deluge. To address this problem, MNP adopts an adaptive sleeping scheme. It reduces idle listening time by putting a node into the sleep state when its neighbors are transmitting its unneeded data. The sleep duration is carefully calculated according to the transmission time.

Approaches to Avoid Broadcast Storms and Hidden Terminals

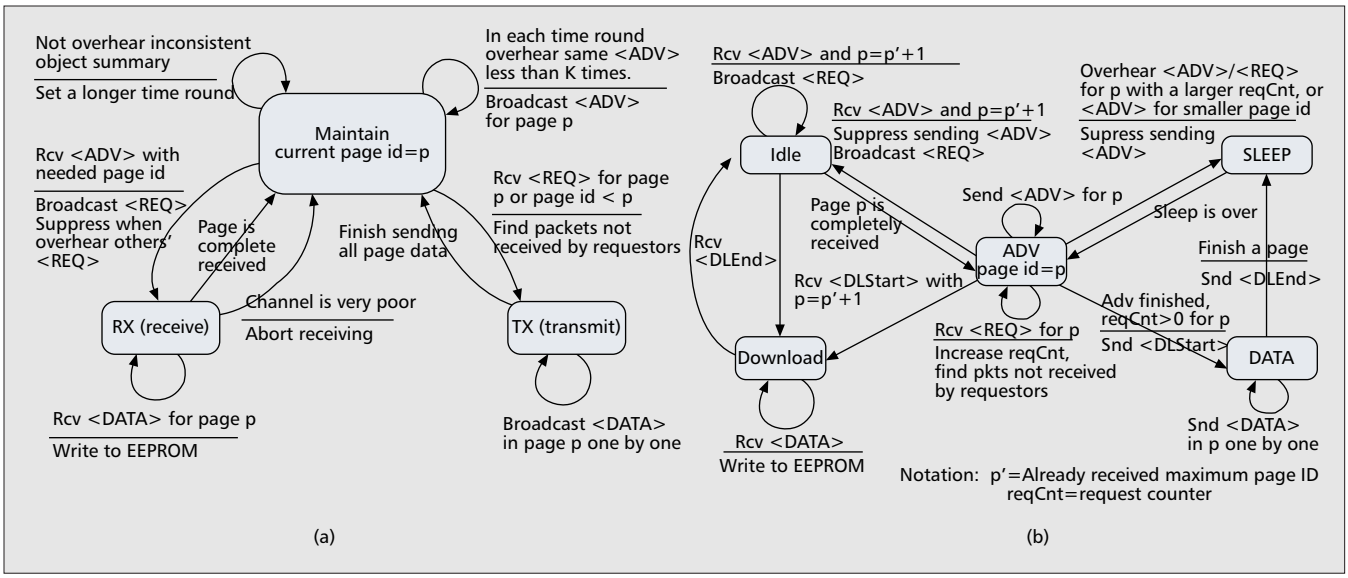
One of the major differences among various reprogramming systems is how they address the broadcast storm and hidden terminal problems. As discussed above, to avoid the broadcast storm problem, data redundancy and sender redundancy need to be overcome.

For data redundancy, two major approaches exist: data aggregation and negotiation. Data aggregation is commonly used to reduce the elastic sensing data. The negotiation-based approach, first proposed in SPIN [3], is used in reprogramming. It introduces three-way handshakes between senders and receivers. A simple negotiation protocol contains three types of messages:

1 ADV: The source node advertises its received objects profile (meta-data).
2 REQ: Its neighbors send back requests after receiving the ADV to notify the source node about which objects are needed.
3 DATA: The source node only sends out the requested objects.

Approaches	Basic Ideas	Systems
Cluster-based [2]	Clusters are formed, and only cluster head nodes are selected to rebroadcast data.	Sprinkler
Counter-based [2]	A counter keeps track of the number of times the same broadcast data has been received. If a node's counter is above a threshold, it will not rebroadcast the data.	Trickle Deluge
Negotiation-based [3]	The priority is given to the sender that is sending a program segment with a smaller segment ID.	SPIN MNP Deluge
Distance-based [2]	Only the node whose distance to the nearest source node is larger than a threshold would rebroadcast the data.	
Location-based [2]	A node uses location information to estimate the additional area that it could cover if it would rebroadcast. It will rebroadcast the program if the estimation is larger than a threshold.	

■ Table 3. *Approaches for reducing sender redundancy of the broadcast storm problem.*



■ Figure 3. *Simplified state diagrams of a) Deluge; b) MNP.*

Popular reprogramming systems (e.g., Deluge and MNP) use the negotiation approach. Through negotiations, the source node knows the requested object (a segment in Deluge and MNP) before sending it out. However, negotiation also adds delays and introduces control overheads. Therefore, the negotiation approach has to be designed and implemented in an efficient and lightweight fashion. We will revisit this problem in later sections.

Table 3 summarizes several approaches to reducing sender redundancy. As a cluster-based approach, Sprinkler divides the whole WSN area into square-shaped clusters, and one node is selected in each cluster as the cluster head. A connected dominating set (CDS) is calculated from the cluster head set. The nodes in CDS will be selected to receive and rebroadcast the data in the first phase in Sprinkler. In the second phase data will be transmitted from CDS nodes to all non-CDS nodes. Compared to the other schemes, the CDS algorithm is centralized and causes extra overhead.

Trickle uses a counter-based approach called polite gossip. Trickle breaks the time into intervals, and at a random time of an interval, a node broadcasts an ADV type message (code summary). If a node has already heard the same ADVs as its own k times in this interval, it "politely" stays quiet. When a node hears an older summary than its own, it broadcasts DATA packets. The number k bounds the number of advertisements made in a given cell by suppressing other nodes, which is adjusted according to the density of the network.

Deluge uses the same polite gossip as in Trickle. However, it adds negotiation (ADV-REQ-DATA) and segmentation on top of Trickle (Fig. 3a). Polite gossip helps Deluge minimize the set of senders to advertise in a time interval. Negotiation further reduces the number of senders by giving higher priority to a node that transmits a segment with a smaller segment ID. In MNP negotiation is also used in a similar way for sender selection (Fig. 3b). Thus, both Deluge and MNP ensure that nodes receive program segments sequentially. Sender selection in MNP also uses a request counter: an advertising node in MNP uses it to keep track of the number of the requests received from its neighbors after advertising a segment. An advertising node will be suppressed if it overhears others' ADVs for the same segment with a larger request count.

TDMA and negotiation are two approaches to overcome the hidden terminal problem. Sprinkler uses TDMA for media access control. Transmitting packets is scheduled in TDMA time slots. The number of TDMA slots affects the latency of the WSN reprogramming service. However, TDMA is not widely supported in current WSNs, and, in addition, using TDMA implies extra network functionality requirements such as time synchronization.

MNP uses negotiation to suppress hidden terminals. The source node sends out the ADV twice. The first ADV contains meta data, and the second ADV adds the number of REQs received after the first ADV. The REQs from the receivers also include this request counter to suppress hidden terminals. Assuming that several nodes are advertising concurrently, if they overhear REQs destined to other nodes, hidden terminals that have a smaller number of requestors will be suppressed.

Reliability Approaches

Program codes should be delivered to the whole network or target nodes reliably. Most existing reprogramming systems use a NACK-based approach since it significantly reduces control traffic, while an ACK-based approach requires an ACK per packet. Error recovery is limited to a single hop in all existing systems since the reliability is decreased exponentially as the number of hops increases.

Deluge and MNP uses REQ as negative acknowledgment (NACK). A REQ packet contains a missing packet bit vector of the advertising segment. Each bit of the vector corresponds to a packet in a segment. Sprinkler unicasts requests from the receiver to get missing packets from source nodes. MOAP has no concept of segmentation, and it uses a sliding window approach to keep track of missing packets.

Scope Selection

Most reprogramming systems only disseminate one program to the whole network. The scope selection function allows administrators or the network to dynamically select any particular nodes to be reprogrammed. An efficient protocol with scope selection should only involve the necessary nodes in forwarding such that it minimizes power consumption and the number of affected nodes during reprogramming, and eventually reprograms all target nodes.

The scope of reprogramming can be as simple as the node ID, group ID, or as complex as application roles, attribute-value pairs with operators on these attributes (e.g. reprogramming all vibration sensor nodes, or all nodes with a temperature reading > 90°F). It is possible that the selection

criteria can only be evaluated at runtime on each individual senor node.

One approach is to begin with a meta-data flooding initiated from the base station. The meta-data carries scope selection criteria to be evaluated on each sensor node. A simple shortest path routing tree is then constructed. Only nodes on the shortest path will act as forwarding nodes in code dissemination.

Aqueduct [12] establishes "aqueducts" of intermediate nodes between source nodes and target nodes. Data is only propagated along these aqueducts. TinyCubus proposes a flexible description language and uses a role concept in scope selection. It reprograms nodes with a particular role via nodes in a specific role (e.g., reprogramming all temperature sensors via vibration sensors). TinyCubus faces a problem because it cannot guarantee that all target nodes can be reached and reprogrammed.

Code Acquisition

Code acquisition is initiated by a node that wants its program to be updated. It tries to get a new program from its peer nodes or the code repository at the base station. A simplified solution is outlined as follows: The node sends out a request containing the object ID. The request is answered by one or more source nodes with the object. A responding source node will be selected as the sender of this reprogramming process based on the received response(s). During the updating process, intermediate forwarding nodes can cache the object in its external storage if possible for future code acquisition requests.

System Behaviors and Evaluations

Evaluation Metrics

Several metrics are commonly used to evaluate a WSN reprogramming system. Among them, reliability, coverage, and autonomy are essential for the correctness of a reprogramming service; therefore, they are generally satisfied in all systems listed in Table 2.

- **Reliability**: Every byte of the program must be correctly received by targeted nodes.
- **Coverage**: All nodes selected in scope selection must eventually be reprogrammed.
- **Autonomy**: Reprogramming should be done with minimal human intervention, and preferably only requires administrators to work on the base station.

In order to achieve practical uses, performance metrics are used to compare different reprogramming systems, which include:

- **Completion time**: Reprogramming will affect the current functioning of a WSN since it generates a large amount of network traffic and demands computation resources. A carefully designed reprogramming system could keep disruption to a minimum, especially when only a partial network is reprogrammed. When disruption is inevitable, reprogramming should be finished as quickly as possible.
- **Energy consumption**: A minimal amount of energy should be used in order to lengthen the total network lifetime. In addition, the energy consumption should be as even as possible among all the nodes to avoid situations where some nodes die too fast.
- **Memory usage**: The program and data memories used by the reprogramming service should be kept to a minimum.

The systems discussed in this article, to some degree, assume different usage models and resource trade-offs. It is very challenging, if not completely impossible, to achieve optimality in all aspects. Trade-offs have to be made to ensure the primary design goal. Therefore, the design and evaluation should always keep its application model in mind.

For example, disseminating small Maté scripts could save more energy than a whole binary code. However, a virtual machine script imposes interpretation overhead at runtime. If a program is only going to execute once, the energy savings from virtual machine encoding will dominate. If it is going to execute hundreds of thousands of times, the energy cost of interpretation could dominate.

When a sensor network is deployed for an emergency scenario that does not require long-term operation (e.g., battlefield intrusion detection, wild fire monitoring), completion time (interruption) is more important than energy saving.

The complexity of reprogramming is directly related to memory usage. For example, adding time synchronization or keeping a neighborhood list could be more efficient in some systems. However, with no dynamic memory allocation, it generates a larger executable image after being compiled. It can be used only if it fits into the memory.

Evaluation Approaches

Real system deployments are desirable to understand system performances in the real world. However, current test developments are usually small scale, containing only dozens of nodes. Simulations are necessary to study large-scale scenarios. TOSSIM is a common simulator to simulate reprogramming systems in TinyOS. TOSSIM simulation uses unmodified reprogramming system codes, and employs a lossy link model to capture the unreliable and asymmetric nature of wireless links. The most commonly used topologies for evaluations are grid networks.

Dynamic Behaviors

Two typical behaviors of segment propagation in MNP and Deluge are shown in Fig. 4. Dynamic behaviors (Figs. 4a–d) [8] can be observed in a network with high density: the segment propagation along the diagonal is significantly slower than that along the edges. Propagations along the edge completely wrap around, which leaves the center area unfilled. A wavefront type of propagation happens in a less dense network (Figs. 4e–h): propagations along the edges and diagonal directions are at comparable rates, with irregularities caused by nonuniform loss rates and contentions. Although not reported in [9], similar behaviors are also observed in MNP. A simple explanation is that there are more neighbors, and thus more contentions, in the center of the network when the node density of a network is high.

Performance Comparisons

Current reprogramming systems adopt various mechanisms as described in pervious sections to improve performance metrics or trade-off one performance metric for another. We briefly summarize the impact of several design factors in this section.

An efficient decoding reduces update size therefore reduces energy consumption and completion time. Incremental Network Programming [5] reports a speedup factor of 9 over XNP for small changes in a new program, and 2∼2.5 for changing a few lines.

MAC also affects the performance. Using TDMA, Sprinkler is 15 times faster than Deluge using CSMA.

With hierarchy, Firecracker achieves threefold speedup, using one-third the transmission cost of Trickle.

To reprogram a partial network, scope selection saves energy over reprogramming the whole network. It shows that in some experiments Aqueduct saves 50 percent data packet with 6 percent extra control overhead compared to Deluge.

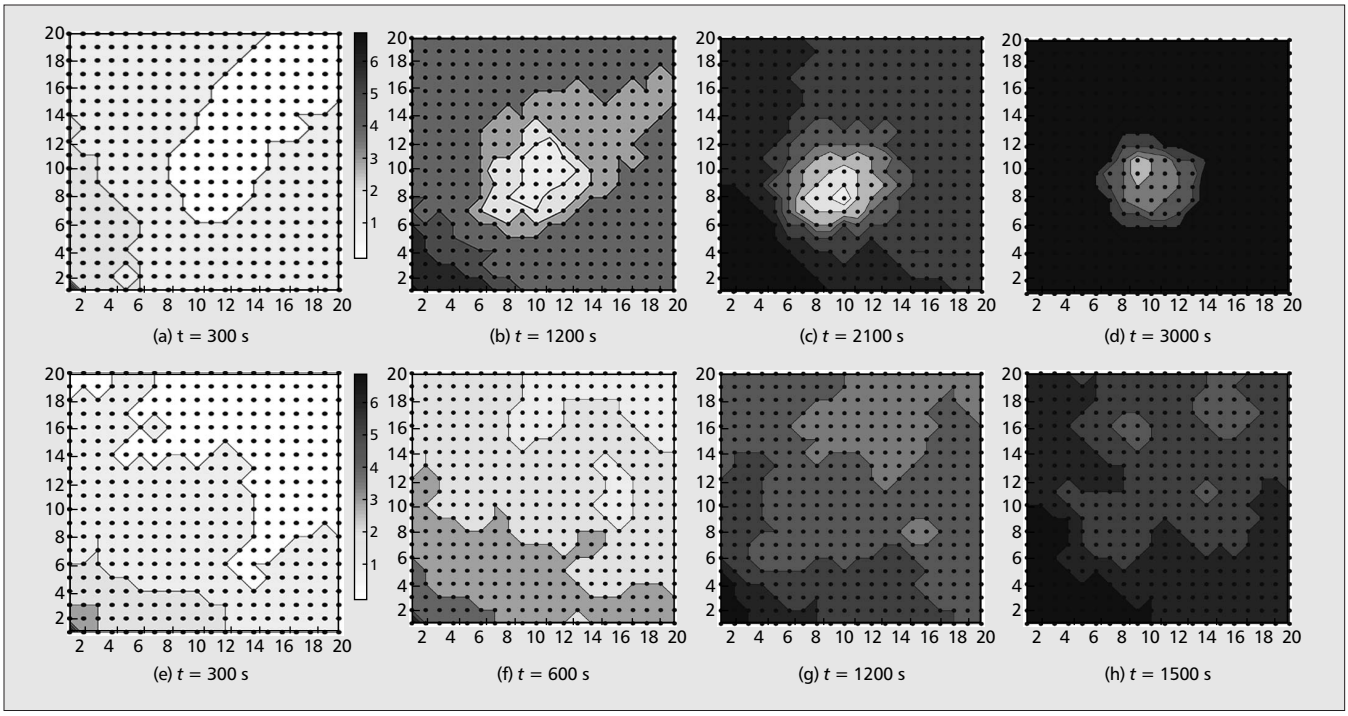


Figure 4. *Dynamic behaviors of MNP in a 20 ×20 grid network. Radio range: 50 ft; node spacing: a–d) 10 ft; e–h) 15 ft.*

Pipelining can decrease the completion time dramatically as observed in Deluge and MNP. Figure 5a simulates the completion time of Deluge and MNP in two lossless networks. Due to the spatial pipelining, when the network scales from 100 nodes (a 10×10 network) to 400 nodes (a 20×20 network), the completion time is significantly faster than without pipelining.

The negotiation scheme reduces the data redundancy and sender redundancy thus the data packets transmitted. It saves energy and decreases completion time. It, however, adds the control overhead at the same time. Figure 5b shows the comparison of data and control packets transmitted in MNP and Deluge. In a lossless network (Fig. 5b), the number of total packets sent in both MNP and Deluge are similar. The result holds when the network scales from 100 to 400 nodes, and from dense networks (node spacing 10 ft) to sparse networks (node spacing 40 ft). MNP, however, finishes reprogramming slower than Deluge, especially when in a dense network. It is largely due to the more complicated negotiation scheme used in MNP.

Segmentation can also affect control overhead dramatically. For a fixed size program, it can be divided into different numbers of segments. Figure 6 shows how segmentation can affect the completion time and number of packets in MNP and Deluge. MNP is sensitive to segmentation while Deluge is not in these simulations. When the number of segments increases, the number of negotiations increases. However, with a smaller number of packets in a segment, it is easier for a receiver to completely receive a segment and then become a source node. The overall performance is affected by which factor dominates.

Concluding Remarks

Reprogramming is important in facilitating the management and maintenance of WSNs, as well as enabling adaptive sensor applications. It becomes a crucial service to the success of

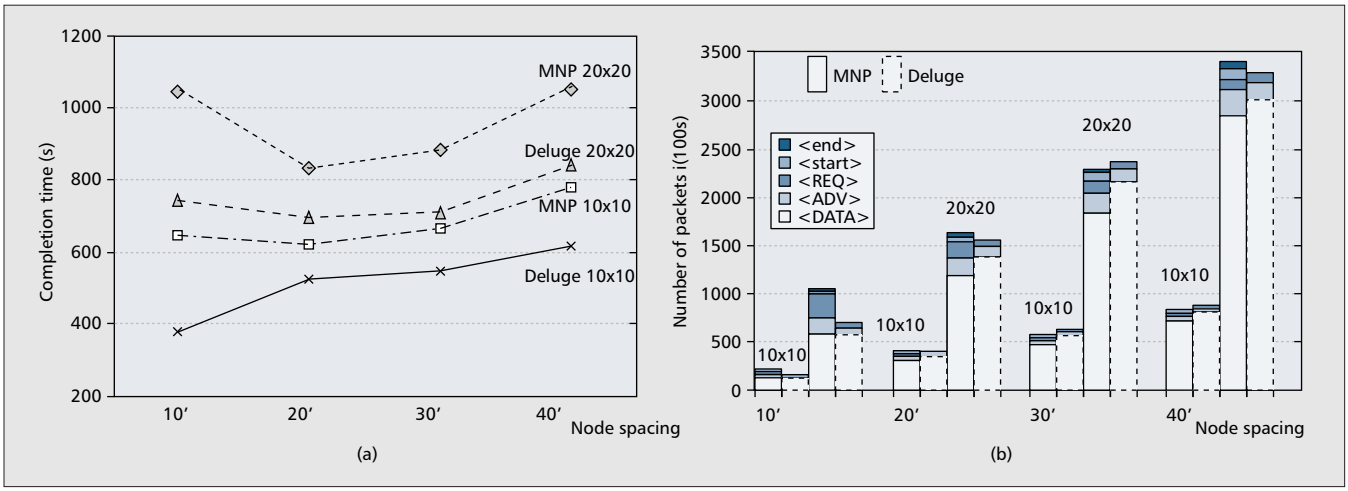
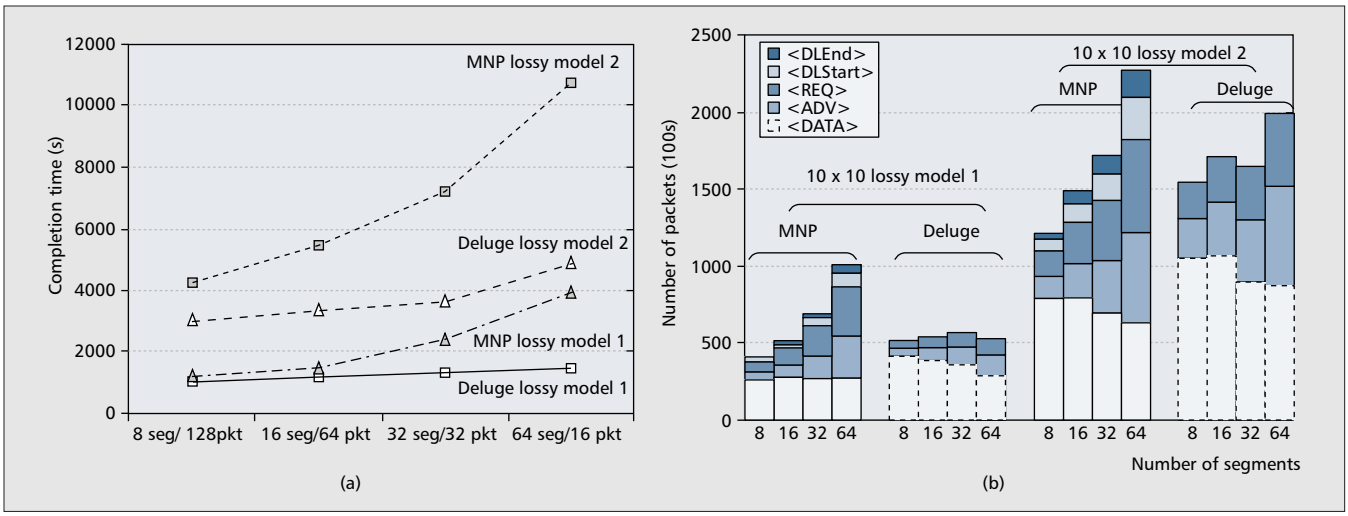


Figure 5. *Eight segments propagation in lossless 10 ×10 and 20 ×20 grid networks: a) completion time; b) transmitted packets. Radio range: 50 ft; node spacing: 10, 20, 30, and 40 ft.*



■ Figure 6. *Segmentation in a 10 ×10 network with two different lossy models: a) completion time; b) transmitted packets. Radio range: 50 ft; node spacing: 10 ft; the same program is divided into 8, 16, 32, and 64 segments.*

WSNs. Currently WSNs are still in the state of active development. In the process of WSN revolution, we will see new hardware platforms (Mica, Telos), new operating systems (TinyOS, SOS), and new applications (well controlled bridge monitoring, randomly deployed wild fire monitoring) keep on emerging. Reprogramming, at the same time, will continue to evolve.

There are lots of open problems that need further investigation to make reprogramming highly usable and efficient. Code dissemination is a continuing focus of current research. However, design trade-offs and impact factors have not been fully understood. Approaches to solving the broadcast storm problem need further study to improve system performance by reducing control overhead. There has been little research on scope selection, complete validation, and code acquisition functions. Design and implementation of energy-efficient routing and one-to-many communication protocols for WSN are still evolving. For practical use, security measures in reprogramming need to be considered (e.g., DoS attacks and node hijacking). Interoperation among heterogeneous network nodes and systems is also important.

Building adaptive WSN applications through reprogramming is a fantastic area. With reprogramming technology advances, we envision that WSNs not only can embed intelligence into environments, but also have embedded intelligence by reprogramming themselves on the fly to dynamic environments.

References

[1] Crossbow Technology, Inc., "Mote In-Network Programming User Reference" and "Mica2 Wireless Measurement System Datasheet," 2003.
[2] S. Y. Ni *et al.*, "The Broadcast Storm Problem in a Mobile Ad Hoc Network," *Proc. IEEE/ACM MOBICOM'99*, pp. 15–62.
[3] J. Kulik, W. R. Heinzelman, and H. Balakrishnan, "Negotiation-Based Protocols for Disseminating Information in Wireless Sensor Networks," *Wireless Networks*, vol. 8 no. 2/3, Mar–May 2002, pp. 169–85.
[4] N. Reijers and K. Langendoen, "Efficient Code Distribution in Wireless Sensor Networks," *Proc. 2nd ACM Int'l. Conf. Wireless Sensor Networks and Apps.*, 2003, pp. 60–67.
[5] J. Jeong and D. Culler, "Incremental Network Programming for Wireless Sensors," *Proc. 1st Annual IEEE ComSoc. Conf. Sensor and Ad Hoc Commun. and Networks (SECON) 2004*, pp. 25–33.
[6] P. Levis *et al.*, "Trickle: A Self-Regulating Algorithm for Code Propagation and Maintenance in Wireless Sensor Networks," *Proc. 1st Symp. Networked Sys. Design and Implementation*, 2004, pp. 15–28.
[7] T. Stathopoulos, J. Heidemann, and D. Estrin, "A Remote Code Update Mechanism for Wireless Sensor Networks," Tech. rep. CENS-TR-30, UCLA, Center for Embedded Networked Computing, Nov. 2003.
[8] J. W. Hui and D. Culler. "The Dynamic Behavior of a Data Dissemination Protocol for Network Programming at Scale," *Proc. ACM SenSys 2004*, pp. 81–94.
[9] S. S. Kulkarni, and L. Wang, "MNP: Multihop Network Reprogramming Service for Sensor Networks," *Proc. IEEE ICDCS 2005*, pp. 7–16.
[10] V. Naik *et al.*, "Sprinkler: A Reliable and Energy Efficient Data Dissemination Service for Wireless Embedded Devices," *26th IEEE Real-Time Sys. Symp.*, Dec. 2005.
[11] P. Levis and D. Culler, "The Firecracker Protocol," *Proc. 11th ACM SIGOPS Euro. Wksp.*, Leuven, Belgium, Sept. 2004.
[12] L. A. Phillips, "Aqueduct: Robust and Efficient Code Propagation in Heterogeneous Wireless Sensor Networks," Master's thesis, Univ. CO, 2005.
[13] P. J. Marrón *et al.*, "Management and Configuration Issues for Sensor Networks," *Int'l. J. Network Mgmt.*, vol. 15, no. 4, July 2005, pp. 235–53.
[14] C. Han *et al.*, "A Dynamic Operating System for Sensor Nodes," *Proc. 3rd Int'l. Conf. Mobile Sys., Apps., and Services*, June 2005, pp. 163–17.

Biographies

QIANG WANG (qiw3@lehigh.edu) is a Ph.D. candidate in the Department of Computer Science and Engineering at Lehigh University. His research interests cover adaptive middleware, wireless sensor network, computer support collaborative work, and software engineering. He was a system analyst and product manager at Huawei Technologies Co. Ltd. from 1996 to 2000. He received a B.S. degree in computer science from Nanjing University, China, in 1996.

YAOYAO ZHU (yaz304@lehigh.edu) is currently a Ph.D. candidate in the Computer Science and Engineering Department at Lehigh University. She received a B.S. degree in electronics from Beijing University, China, in 1995 and an M.S. degree in computer engineering from the University of Cincinnati in 2001. Her research interests include wireless sensor networks, adaptive middleware, and machine learning.

LIANG CHENG (cheng@cse.lehigh.edu) is director of the Laboratory Of Networking Group (LONGLAB, http://long.cse.lehigh.edu) and an assistant professor in the Computer Science and Engineering Department at Lehigh University (http://www.cse.lehigh.edu/~cheng/). He received his Ph.D. degree from Rutgers University, New Jersey. He has been the principal investigator (PI) and co-PI of projects funded by NSF and DARPA. He is an awardee of the Christian R. & Mary F. Lindback Foundation Minority Junior Faculty Award.