

MCP: an Energy-Efficient Code Distribution Protocol for Multi-Application WSNs

Weijia Li, Youtao Zhang, Bruce Childers

Computer Science Department, University of Pittsburgh, Pittsburgh, PA 15260

Abstract. In this paper, we study the code distribution problem in multi-application wireless sensor networks (MA-WSNs), i.e., sensor networks that can support multiple applications. While MA-WSNs have many advantages over traditional WSNs, they tend to require frequent code movements in the network, and thus here new challenges for designing energy efficient code dissemination protocols.

We propose MCP, a stateful Multicast based Code redistribution Protocol for achieving energy efficiency. Each node in MCP maintains a small table to record the interesting information of known applications. The table enables sending out multicast-based code dissemination requests such that only a subset of neighboring sensors contribute to code dissemination. Compared to broadcasting based schemes, MCP greatly reduces signal collision and saves both the dissemination time and reduces the number of dissemination messages. Our experiments results show that MCP can reduce dissemination time by 25% and message overhead by 20% under various network settings.

1 Introduction

Wireless sensor networks (WSNs) have recently emerged as a promising computing platform for applications in non-traditional environments, such as deep sea and wild fields. They are usually under tight memory and energy constraints, e.g., a Telos sensor node has only 48KB program memory [15]. Many WSNs can load only one application per node and the same application to all nodes in the network. We refer to these WSNs as single-application WSNs or SA-WSNs.

While one sensor is usually small and cheap, as the network size scales, a large WSN may consist of thousands of sensors making it economically less appealing to run just one application. Recently, researchers have envisioned the wide adoption of multi-application WSNs or MA-WSNs, which can support several applications in one network infrastructure [20, 17]. In a MA-WSN, a sensor stores the code of multiple applications in its external flash memory and loads the selected application into its program memory for the desired functionality. The latter program is referred to as the active application. The recent technology and research advances clear ways for adopting MA-WSN design. Firstly it is now possible to integrate more memory (especially flash memory) with the same budget [19]; secondly different sensing, clustering, routing and data aggregation

protocols have been proposed to reduce the energy consumption of performing functionalities of one application [1]. As a result, it is now possible to support multiple applications during the lifetime of a MA-WSN.

MA-WSNs have many advantages over SA-WSNs. For example, MA-WSN can be deployed in a national park to monitor both wildfires and animal movement. More sensors could be set to monitor the animal movement in the early morning or late afternoon when animals tend to leave from or return to their habitats; while more sensors could be set to monitor wildfires in the summer season when the weather is dry and the chance to catch fires is high. By exploiting the same network infrastructure for both events, (1) MA-WSNs save the investment and effort in deploying and testing two sensor networks; (2) the sensor network adapts better to the dynamic changing environment and even adjusts the coverage according to the need.

However, not all nodes in MA-WSNs have the code of all running applications. Some sensors may need to switch to run the code that can be found neither in their flash memory nor from their neighboring sensors. This results in more code movements and makes it critical to design energy-efficient post-deployment code dissemination/redistribution protocols in MA-WSNs. Most existing code dissemination protocols, such as Deluge [5], MNP [6], Stream [14] are designed to disseminate the same code to all sensors in the network. While it is possible to adopt a naive *drop-after-receive* scheme that discards unnecessary code after dissemination, applying these protocols in MA-WSNs tends to introduce high overhead. A recent proposed protocol, Melete [20], has a similar design goal. However, it employs broadcast strategy in dissemination, which introduces significant signal collision and communication overhead in disseminating applications with large code sizes. MDeluge [21] uses a tree-based routing approach to disseminate the application code to the sensors that need to run the application. The pre-computed routing table is fixed during one application upgrade, thus it is subject to the network congestion and single point failure.

In this paper, we propose a multicast-based code redistribution protocol, MCP, for the purpose of achieving energy efficiency. MCP employs a gossip-based source node discovery strategy. Each sensor summarizes the application information from overheard advertisement messages. Future dissemination requests are forwarded to nearby source nodes rather than flooding the network. Our experiments show that MCP greatly reduces both dissemination time and message overhead, and achieves 10-20% reductions in various settings.

In the remainder of the paper, we describe the code dissemination background in Section 2 and the MCP protocol in Section 3. Section 4 evaluates the effectiveness of MCP under different settings. Related work is discussed in Section 5, and Section 6 concludes the paper.

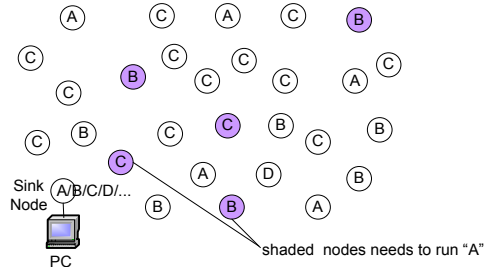


Fig. 1. A multi-application WSN (MA-WSN).

2 Code Dissemination and Problem Statement

2.1 Code Dissemination

As shown in Fig. 1, a WSN consists of one sink node and a number of sensors, e.g. MICA2 nodes. The sink node connects directly to the PC and thus has no power and computation restrictions. Each MICA2 node has 128KB program memory, 4KB data memory and 512KB external flash memory.

A remote sensor in MA-WSN keeps the code images of several applications in its external flash and loads the active application to its program memory. To execute a different application, the bootloader of the sensor swaps in the desired code image and reboots the sensor. A recent study showed that to support effective dissemination, the whole code dissemination functionality or at least part of it should be integrated in the active application [14]. Thus, a sensor normally performs the operations of the active application and enters the code dissemination mode only after receiving special primitives, i.e., data packets with special opcode bits.

An application to be disseminated is usually divided into a sequence of pages, each of which consists of multiple data packets (Fig. 2). In TinyOS each packet is 29 bytes long and contains a 23 byte payload. To adapt to the lossy wireless communication channel, Deluge [5] disseminates code at page granularity in increasing order of page number. That is, a requesting sensor has to finish receiving all the packets in one page before sending out requests for the next page; however, packets within one page may be received out of order as some packets may be lost and need to be retransmitted. During code dissemination, the requesting sensor buffers packets from the current page in data memory and writes to the external flash after receiving the whole page.

2.2 Problem Statement

The problem that we study in this paper is to design an energy-efficient code dissemination protocol in MA-WSNs. We illustrate the protocol design challenges using the following example. As shown in Fig. 1, three applications

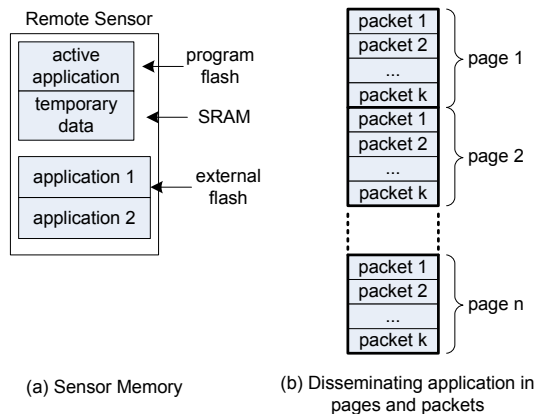


Fig. 2. Sensor memory organization and code dissemination in pages and packets.

are distributed across different nodes in a network. A problem arises when there is a need to reprogram some nodes to run application A.

There are two existing approaches. A naive solution is to directly apply Deluge and disseminate application A from the sink to all sensors. After dissemination, the nodes that do not need A discard the code from their storage. The solution is clearly not a good choice due to unnecessary packets transmissions to the nodes that don't need it. The other solution is to let requesting nodes initiate code dissemination and fetch A from nearby sensors. Melete [20] is such a protocol — the nodes that need to run A broadcast their requests within a controlled range and discover the source nodes that have A. Sources then send back the requested data packets. However, as a stateless protocol, Melete does not record the source nodes and has to discover them repeatedly. When transmitting applications with multiple pages, multiple sources within the range may respond and thus create significant signal collision.

In this paper, our goal is to design an energy-efficient code dissemination protocol for MA-WSNs. Our target is to reduce both dissemination completion time and the number of messages transmitted during dissemination.

3 The MCP Protocol

3.1 Overview

An overview of our protocol is as follows.

- Sensors in MCP periodically broadcast **ADV** messages to advertise their knowledge about running applications in the network, which is similar to Deluge. Each sensor summarizes its overheard **ADV** messages in an *application information table (AIT)*.

- To reprogram a subset of sensors, the sink floods a dissemination command that guides which sensors should switch to run application A. For example, a command “[B→A, p=0.25]” indicates that the sensors whose active application is “B” should switch to “A” with a 25% probability. That is 25% of the nodes that are currently running application “B” will switch to “A”.
- After receiving the command from the sink, each sensor identifies its dissemination role as one of the followings.
 - (i) a *source* if the sensor has the binary of application A;
 - (ii) a *requester* if the sensor does not have the binary of A but needs to switch to run A; or
 - (iii) a *forwarder* if the sensor is neither a *source* nor a *requester*.
- A *requester* periodically sends out requests (i.e., REQ messages) to its closest source, until it acquires all the pages of application A. Instead of broadcast, the REQ messages are sent to the source via multicast. A requester resends the REQ message until it timeouts. It tries to request data from each source node several times before marking the node as a *temporary non-available* source.
- A source node responds with the data (i.e., Data messages) that contain code fragments while a forwarder forwards both request and data packets.

Similar to Melete and Deluge, MCP has three types of messages: an ADV message that advertises interesting applications; a REQ message that asks for packets of a particular page; and a Data message that contains one data packet (i.e, a piece of code segment).

3.2 ADV Message and Application Information Table (AIT)

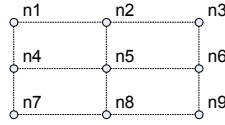
In MCP, each sensor periodically broadcasts ADV messages, and summarizes the information of overheard ADV messages into a small application information table (AIT). Fig. 3 illustrates the algorithm.

Each ADV message contains the information of one application: (i) an application ID and version number; (ii) the number of pages of the application; (iii) the information of two closest source sensors — the source ID and number of hops to the source (S, H); (iv) the CRC checksum. If a sensor has multiple known applications, it advertises them in a round-robin fashion. Note that a sensor may not have the code images of all its known applications.

The AIT summarizes the overheard ADV messages. In addition to the application summary, AIT stores up to three closest source nodes for each known application, and the uplink sensor ID for each source, i.e., from which the source information was received. The size of each application entry in the AIT is 12 bytes. Assume that the number of the applications running in the network is 10, the AIT size will be only 120 bytes, which makes it fit perfectly in the program memory.

When an incoming ADV message contains new information, the corresponding entry in the AIT is updated. Assume a sensor S1 receives an ADV message from S2, and the message identifies two nearby sources (S3, H3) and (S4, H4) where H3 and H4 indicate the number of hops from S2 to sources S3 and S4. If

Network: Assume n1 has A1; n3, n5 n9 will change to A1



On Node N9:

Application ID	version	# pages	node ID	hop #	uplink ID
A1	1	8	n1	4	n8
			n3	2	n6
			n5	2	n8
A2	1	8
		
		

On Node N4:

Application ID	version	# pages	node ID	hop #	uplink ID
A1	1	8	n1	1	n1
			-	-	-
			-	-	-

Fig. 3. Application Information Table.

S1 already records the information of three sources (S5, H5, U5), (S6, H6, U6), and (S7, H7, U7), then it updates the AIT table according to the following rules.

- If one entry in AIT table records the previous message from the same uplink S2 and it refers to the same source, e.g. S5=S3 and U5=S2, then the information in the ADV message represents the up-to-date source information and replaces the old entry.
- If one entry in the AIT records a longer path to an advertised source, e.g. S5=S3, U5≠S2, and H5>(H3+1), then the hop count and uplink node from the ADV message replace those in the AIT.
- If the advertised source cannot be found in the AIT, and there is an invalid entry in the table, then the new source is inserted into the table.
- If the ADV message advertises a closer source than one of those in the AIT table, then the closer source replaces the farthest source in the AIT.

Each sensor advertises the application in the AIT in a round-robin fashion, and prioritizes the applications whose entries have been recently updated: (i) the applications whose sources were recently updated are advertised before those that were not; (ii) in one round, the applications whose sources were recently updated are advertised three times while others are advertised once. In addition to normal ADV advertisement, an application is advertised if the sensor receives a broadcast request for that application, as we elaborate next.

3.3 Request Multicasting

In MCP, a requester continues to send out request messages until it receives all pages of the target application. Given the target application, the requester searches the AIT for a closeby live source and constructs a REQ message as follows

$$\text{REQ} = [\text{S}, \text{H}, \text{pgNum}, \text{bv}]$$

where S indicates the selected source node, H indicates the maximum number of hops that the message may travel, pgNum and bv indicate the current working page and the requested packets in the page. If the AIT records more than one source node, then the requester selects the closest live source and sets H to $h+\delta$ where h is the number of hops to S (recorded in the AIT), and δ is the hop count slack allowed in the dissemination. Fig. 4 illustrates the involved nodes when $h=2$ and $\delta=1$. These nodes routed through a gradient-based region [1] to the source.

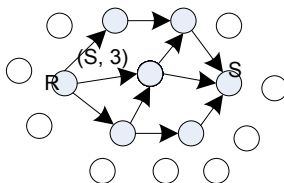


Fig. 4. Gradient-based request routing (R and S are requester and source nodes respectively; $h=2$; $\delta=1$).

A requester continues sending the REQ messages when it can not finish the page before timeout. After several tries, it marks the source that it tried to reach as an *unreachable* source. The number of tries varies based on the distance to the source.

If the AIT does not record any nearby source, then the requester sets S to be *null*, indicating the REQ message is sent to all neighbors. After receiving a broadcast request, an idle forwarder forwards the request unless the message has travelled the maximum number of hops; an idle source node always responds with requested packets.

Since each requester sends out REQ messages independently, different requesters may work on different pages. MCP allows node preemption. If a REQ message asking for page x reaches a working node who is currently working on page y , and $x+1 < y$, then the node quits the current state and switches to serve the request. If the node is a forwarder, then it forwards the request; if the node is a requester or a source, then it must have the requested page and thus will respond with the requested packets. The node enters the idle state after serving the request.

3.4 Caching

During code dissemination, some requesters or forwarders, while working on the current page, may overhear packets from pages with larger indices. As code pages are requested strictly in increasing order, a requester will work on large-number-indexed pages, and a forwarder has a high possibility to receive requests for these pages.

To improve transmission efficiency, sensors in MCP buffer such packets in their data memory. The space that can be dedicated to caching on a wireless sensor is usually very limited. While it is possible to exploit external flash for caching, accessing external flash is slow and writing it has to be performed in 256-byte blocks, which complicates the design and wastes the energy.

Caching on a requester is straightforward as the sensor always caches the next several pages in addition to the current working page. However, it is slightly more complicated on a forwarder node as it gets requests from different requesters that work on different pages and may suffer from thrashing if it takes turns to serve these requests. In MCP, a forwarder gives priority to pages with smaller indices. We set a timer for the cached page and clear the page after serving a request or timeout.

4 Experiments

4.1 Settings

We implemented MCP on the TinyOS [18] platform. For comparison, we also implemented Melete [20] and studied various network settings using TOSSIM [9].

We simulated mesh MA-WSNs of different sizes. We set the default spacing factor to 15 and model the lossy communication channel using the tool provided by TinyOS. There are four applications each of which is uniformly distributed across the network. In the default setting, 30% of the sensors have application A and there is a request from the sink to reprogram 20% of the other sensors to run A. MCP disseminates the code from in-network sources instead of the sink.

4.2 Message overhead

Fig. 5 shows the breakdown of the number of messages with different dissemination protocols. Without considering advertisement messages, Melete and Deluge have about the same message overhead, which was also reported in [20]. There are a large number of ADV messages in Deluge, and a negligible number in Melete. The reason of such difference is Deluge depends heavily on incoming ADV message, e.g., a sensor node only sends out new requests if it receives ADV messages indicating its neighbors have more up-to-date data. Instead, in Melete, requesters receive the command from the sink code and then know the target application and its size. The requesters can proactively send out more requests after timeouts or receiving one complete page. The ADV messages contribute to 37-40% of the total message overhead in Deluge.

Our scheme takes a similar approach as Melete but requires some ADV messages to update the AIT before, during and after the code switch. The ADV's

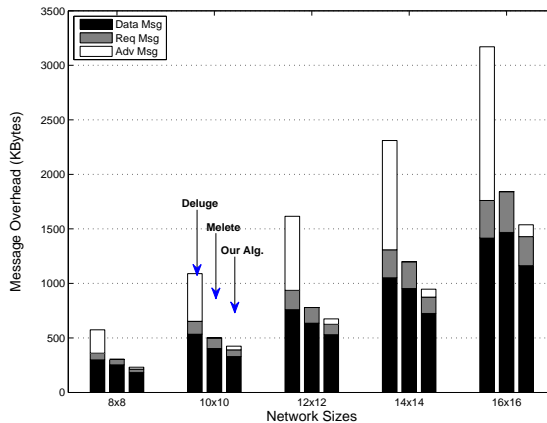


Fig. 5. Message overhead.

overhead is low compared to the request and data transfer message overhead. On average our scheme reduces about 20% of the message overhead from Melete. The main reason for this reduction is that Melete tends to have multiple responders within a small range and has a higher possibility of signal collision. MCP alleviates the problem by choosing one closeby source, which reduces the number of data packets in transmission.

4.3 Completion time

Fig. 6 compares the dissemination completion time of the different protocols. For the Deluge result, we record the time interval used by all requesters to complete the new code downloading. In practice, the Deluge protocol may still proceed to flood all sensors since it is not designed to update a subset of sensors. MCP requires less time to finish dissemination; on average it saves 45% and 25% over Deluge and Melete respectively.

4.4 Sensitivity to Node Distribution

Fig. 7 illustrates message overhead with a different number of sources and requesters. We omit the dissemination time figure which exhibits similar results. Along the X axis, (a,b) denotes that out of all the sensor nodes, a% sources and b% requesters are randomly selected in the field. We observed that the overhead tends to increase with more requesters and fewer sources. The difference is not significant.

Fig. 8 compares the message overhead when sources and requesters are distributed with location concentration. **EvenD** denotes that all nodes are evenly distributed. **CornerD** denotes that sources and requesters are distributed at the

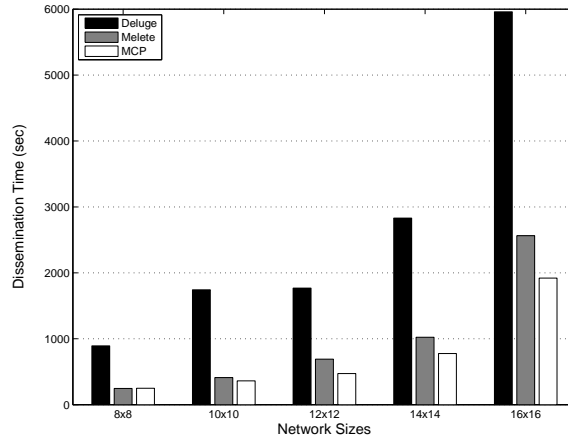


Fig. 6. Dissemination Time.

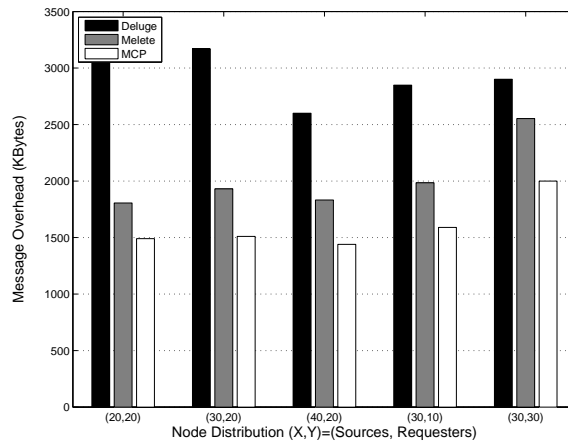


Fig. 7. Dissemination with Different Number of Sources and Requesters.

two diagonal corners of the rectangle field. **SideD** denotes that sources and requesters are distributed along two sides of the field. From the figure, Melete has better performance than Deluge under even distribution. However, it generates significant conflicts and performs worse than Deluge when the nodes are unevenly deployed. MCP has consistently better results over Melete and Deluge. For the corner and side settings, MCP and Deluge are similar as almost all nodes are involved in the dissemination.

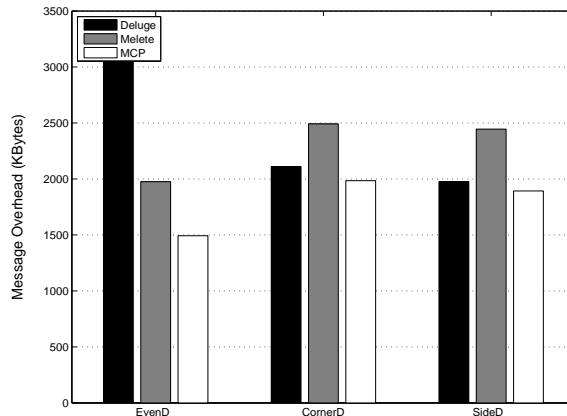


Fig. 8. Dissemination with Uneven Source/Requester Node Distribution.

4.5 Sensitivity to Application Sizes

Fig. 9 shows message overhead with different application sizes. Due to the epidemic dissemination, Deluge exhibits approximately linear message overhead when increasing the application size from 8 to 16 pages. Both Melete and MCP greatly reduce the communication overhead; however, they have slightly more than linear message overhead due to independent page requesting from requesters. MCP has a nearly constant message overhead reduction versus Melete, varying from 17.5% for 8 pages to 18.1% for 16 pages.

4.6 Sensitivity to Cache Sizes

Fig. 10 summarizes message overhead of Melete and MCP with different cache sizes, i.e., the number of code pages that may be cached in memory. Here $N=1$ denotes that there is no caching. From the figure, MCP achieves significant communication overhead reduction when caching one or two future pages, and diminishing benefits when with larger cache sizes. The reason is that in MCP, a request message can preempt a working node (a source, a requester, or a forwarder) if that node works on a page with a larger page number and the page index difference is bigger than one. In this way, MCP prioritizes slow requesters such that they can keep up the pace with the nearby dissemination and take advantage of cached packets on the neighboring sensors.

5 Related Work

Since sensors are left unattended after deployment, post-deployment code dissemination is an important topic in designing wireless sensor networks. Besides Deluge [5] and Melete [20] that we have discussed in the paper, many other protocols have been proposed. MNP [6] and Trickle [10] are protocols designed in

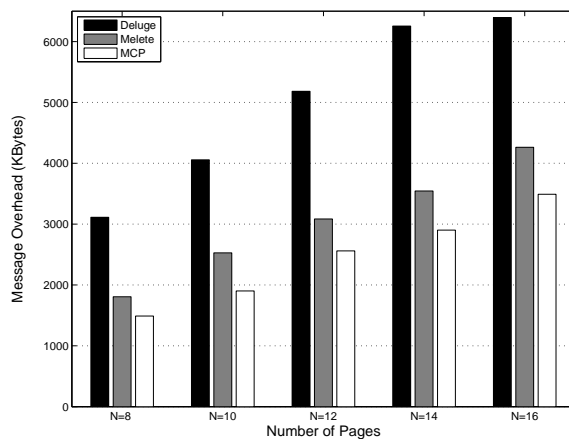


Fig. 9. Dissemination with Different Number of Pages.

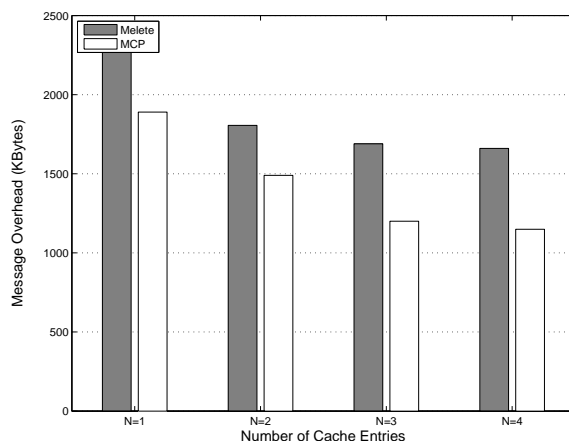


Fig. 10. Dissemination with Different Cache Sizes.

TinyOS to support multihop code dissemination. MDeluge [21] uses Deluge [5] to manage code dissemination for MA-WSNs; however, it uses a fixed routing table, and is subject to the network congestion and single point failure. Infuse [7] is a TDMA-based dissemination protocol. Impala/ZebraNet [13] provides a middleware layer to support code update. These protocols propagate the desired code to all sensors in the network.

Reducing the amount of data transferred during dissemination is an effective approach to reduce overhead. [16] and [4] proposed to generate and propagate a *diff* script instead of the complete binary code image. [12] performed update

conscious register allocation and data allocation to reduce the script size. The code size can also be reduced by disseminating modules with symbolic names and performing remote linking before execution [2], or by disseminating virtual machine primitives [11].

To defend against security attacks during code dissemination, [8] integrated digital signature and hashing chaining mechanisms to ensure page-level data integrity. Since packets within one page may arrive out of order, security attacks within one page is still possible. [3] performs packet level security checks to defend such attacks with the tradeoff to enforce a stronger dissemination order within page.

6 Conclusions

In this paper, we propose a multicast-based code dissemination protocol, called MCP, for efficient code dissemination in MA-WSNs. In MCP, each sensor summarizes overheard information of nearby sources in a small table such that its dissemination requests can be multicasted to selected source. Compared to design that floods requests to all neighboring sensors, MCP significantly reduces signal conflicts. Our experimental results show that MCP can reduce dissemination time by 25% and message overhead by 20% on average.

7 Acknowledgement

This work is supported in part by NSF under grant CCF-0641177, CNS-0720595, CCF-0811352, CCF-0811295, CNS-0720483, CCF-0702236, and CNS-0551492.

References

1. M. Chu, H. Haussecker, and F. Zhao, "Scalable Information-Driven Sensor Querying and Routing for ad hoc Heterogeneous Sensor Networks," In *International Journal on High Performance Computing Applications*, 16(3):90-110, Fall 2002.
2. A. Dunkels, N. Finne, J. Eriksson, and T. Voigt, "Run-Time Dynamic Linking for Reprogramming Wireless Sensor Networks," *ACM International Conference on Embedded Networked Sensor Systems (SenSys)*, pages 15–28, 2006.
3. P.K. Dutta, J.W. Hui, D.C. Chu, and D.E. Culler, "Securing the Deluge Network Programming System," *International Symposium on Information Processing in Sensor Networks (IPSN)*, pages 326–333, 2006.
4. J. Jeong, and D.E. Culler, "Incremental Network Programming for Wireless Sensors," *IEEE Sensor and Ad Hoc Communications and Networks (SECON)*, pages 25–33, 2004.
5. J. W. Hui and D. Culler, "The Dynamic Behavior of a Data Dissemination Protocol for Network Programming at Scale," In *International Conference on Embedded networked sensor systems (SenSys)*, pages 81-94, 2004.
6. S.S. Kulkarni, L. Wang, "MNP: Multihop Network Reprogramming Service for Sensor Networks," In *IEEE International Conference on Distributed Computing Systems*, 2005.

7. S.S. Kulkarni, and M. Arumugam, "Infuse: A TDMA Based Data Dissemination Protocol for Sensor Networks," In *Conference on Embedded Networked Sensor Systems*, 2004.
8. P.E. Lanigan, R. Gandhi, and P. Narasimhan, "Sluice: Secure Dissemination of Code Updates in Sensor Networks," In *the 26th Intl. Conference on Distributed Computing Systems*, 2006.
9. P. Levis, N. Lee, M. Welsh, and D. Culler, "TOSSIM: Accurate and Scalable Simulation of Entire TinyOS Applications," In *International Conference on Embedded networked sensor systems (SenSys)*, 2003.
10. P. Levis, N. Patel, S. Shenker, and D. Culler, "Trickle: A Self-regulating Algorithm for Code Propagation and Maintenance in Wireless Sensor Networks," In *USENIX/ACM Symposium on Networked Systems Design and Implementation (NSDI)*, 2004.
11. P. Levis, and D. Culler, "Mate: A Tiny Virtual Machine for Sensor Networks," *International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS)*, pages 85–95, 2002.
12. W. Li, Y. Zhang, J. Yang, and J. Zheng, "UCC: Update-Conscious Compilation for Energy Efficiency in Wireless Sensor Networks," In *ACM Programming Languages Design and Implementation (PLDI)*, 2007.
13. T. Liu, C.M. Sadler, P. Zhang, and M. Martonosi, "Implementing Software on Resource-Constrained Mobile Sensors: Experiences with Impala and ZebraNet," In *International Conference on Mobile Systems, Applications, and Services*, 2004.
14. R.K. Panta, I. Khalil, and S. Bagchi, "Stream: Low Overhead Wireless Reprogramming for Sensor Networks," *IEEE Conference on Computer Communications (Infocom)*, 2007.
15. J. Polastre, R. Szewczyk, and D. Culler, "Telos: Enabling Ultra-Low Power Wireless Research," *IPSN'05*, pages 364-369, 2005.
16. N. Reijers, K. Langendoen, "Efficient Code Distribution in Wireless Sensor Networks," In *ACM Workshop on Wireless Sensor Networks and Applications (WSNA)*, 2003.
17. J. Steffan, L. Fiege, M. Cilia, A. Buchman, "Towards Multi-Purpose Wireless Sensor Networks," In *the 2005 Systems Communications*, pages 336-341, 2005.
18. TinyOS website. <http://www.tinyos.net/>
19. MICAz Wireless Measurement System. <http://www.xbow.com/>.
20. Y. Yu, L. J. Rittle, V. Bhandari, and J. B. LeBrun, "Supporting Concurrent Applications in Wireless Sensor Networks," In *International Conference on Embedded networked sensor systems (SenSys)*, pages 139-152, 2006.
21. X. Zheng, B. Sarikaya, "Code Dissemination in Sensor Networks with MDeluge," In *Sensor and Ad Hoc Communications and Networks (Secon)*, pages 661-666, 2006.