软件学报 ISSN 1000-9825, CODEN RUXUEW
Journal of Software,2011,22(Suppl.(1)):175−181

E-mail: jos@iscas.ac.cn
http://www.jos.org.cn
Tel/Fax: +86-10-62562563

EasiLWR:一种轻量级传感器网络无线重编程方法*

邱杰凡[1,2+], 李 栋[1], 石海龙[1,2], 崔 莉[1]

[1](中国科学院 计算技术研究所,北京 100190)

[2](中国科学院 研究生院,北京 100049)

EasiLWR: A Lightweight Wireless Reprogramming Approach for Sensor Network

QIU Jie-Fan[1,2+], LI Dong[1], SHI Hai-Long[1,2], CUI Li[1]

[1](Institute of Computing Technology, The Chinese Academy of Sciences, Beijing 100190, China)

[2](Graduate University, The Chinese Academy of Sciences, Beijing 100049, China)

+ Corresponding author: E-mail: qiujiefan@ict.ac.cn

Qiu JF, Li D, Shi HL, Cui L. EasiLWR: A lightweight wireless reprogramming approach for sensor network. *Journal of Software*, 2011,22(Suppl.(1)):175−181. http://www.jos.org.cn/1000-9825/11019.htm

Abstract: Wireless reprogramming approach of sensor node is a requirement for flexible configuration and update. Considering the communication overhead and the program overhead, this paper presents a novel lightweight wireless reprogramming approach – EasiLWR with the following merits. By comparing the functions of the new program and the old program, the different codes between functions are transferred instead of entire codes. The communication overhead is reduced. Moreover, using the code shift, the parts of program, which need frequently update are stored and implemented in RAM rather than in the internal Flash of microcontroller unit (MCU). This translates to a reduced programming time. The experimental results show the outperformance of EasiLWR.

Key words: sensor network; wireless reprogramming; difference code of function; code shift

摘 要: 无线重编程适用于各种应用场景下对传感器节点进行灵活的配置和升级.考虑到无线重编程会带来很大的通信开销和更新开销,提出了一种新的轻量级无线编程方法——EasiLWR.它通过函数级差异对比,计算新老代码中存在的差异代码.在无线传输时,仅传输差异代码,大幅减少了需要传输的代码量,有效降低了通信开销.它利用代码转移将部分需要频繁升级的代码存储在RAM中并加以执行,当进行重编程时,只需要对RAM进行写操作.从而避免了对 Flash 的写操作,缩短了更新时间,有效降低了更新开销.实验结果表明,与现有无线重编程方法相比,EasiLWR在降低通信开销及更新开销方面优势显著.

关键词: 传感器网络;无线重编程;函数级差异代码;代码转移

由于无线传感器节点自身资源的限制,同时受到周围环境变化的影响,开发者很难在编程阶段考虑到节点在部署后所有可能遇到的意外情况以及可能的需求.特别是当节点被部署在无人值守区域或者网络规模较为庞大时,也不可能通过预先测试获得这些信息.这时,无线重编程便成为解决上述困难的有效途径.

* 基金项目: 国家自然科学基金(61003293); 国家科技重大专项(2010ZX03006-003-02); 中国科学院计算技术研究所知识创新项目(20106030)

收稿时间: 2011-05-02; 定稿时间: 2011-07-29

然而现有无线重编程方法存在着诸多问题:第一,由于重编程时,传输的代码量较大,会造成传输时间长、通信开销大、信道被长期阻塞等问题;第二,节点在重编程期间需要进入特殊的更新状态,无法响应正常的操作,从而使整个网络处于不稳定状态;第三,某些使用内部程序 Flash 存储代码的嵌入式芯片,可能由于电池电压过低导致写 Flash 操作失败[1],无法完成重编程.

本文针对当前无线重编程方法存在的以上不足,提出了一种新的轻量级无线重编程方法 EasiLWR.它采用函数级差异对比技术,在计算新旧代码间的差异代码的同时保留了程序的结构信息.在对节点更新过程中只需传输差异代码,大大降低了通信开销.另一方面,代码写入 Flash 的时间要远远超过写入 RAM 的单位时间,为了提高更新效率,缩短更新时间,EasiLWR 通过代码转移,实现了在 RAM 中存储并执行代码.同时由于避免了对 Flash 的写操作,可以有效延长可重编程时间.实验结果表明,EasiLWR 的通信开销和更新时间较 Elon,Deluge 和 Hermes 等现有无线重编程方法具有明显优势.

1 相关工作

目前,国内外大量研究人员对适用于传感器网络中的无线重编程方法进行了深入的研究.Deluge[2]是较早提出的无线重编程方法.该方法在代码分发阶段需要传输重编程协议和整个 TinyOS 镜像.由于这部分代码的尺寸往往超过嵌入式芯片内部 Flash 的大小,待升级节点通常将这部分代码保存在外部 Flash 中.当需要重编程时,通过 Bootloader 将代码读入内部 Flash,并在硬件重启后完成更新.针对 Deluge 的不足,Elon [3]以 TinyOS 为基础,将代码以事件(event)为单位放入 RAM 中执行,极大地提高了更新效率,但是它在重编程时仍以整个差异函数作为传输的基本单位,包含了大量的无关代码.

增量式重编程方法[4−8]通过传输差异代码达到降低通信开销的目的.其中 Hermes[6]使用 Rsync 算法计算字节级差异,可以大幅降低通信开销.但是,由于它必须对外部 Flash 进行读写操作,因而需要较长的更新时间.Koshy 和 Pandey[5]试图通过给每个函数末尾添加溢出空间(slop region)来存放插入的代码,尽量避免代码重建.但是溢出空间会导致大量无效的存储碎片,而且插入的代码量受到溢出空间大小的限制.

除了以上专门的无线重编程方法之外,SOS[9],Contiki[10]等传感器网络操作系统使用了动态链接技术实现重编程,但是这些操作系统在重编程时需要传输符号表和重定位表,增加了通信开销,并且无法对系统内核模块进行重编程.Mate[11]和 ASVM[12]在节点上实现了 Java 虚拟机技术,可以在传输少量代码的情况下完成重编程.然而 Java 虚拟机代码是一种紧凑型代码(compact code),与原生码(native code)相比,执行效率过低,并且表达能力有限.

2 研究背景

嵌入式操作系统 TinyOS[13]被广泛应用于无线传感器网络中,当前大多数无线重编程方法都是以它作为研究的基础.在本文中我们也以 TinyOS 作为研究基础,并以 TelosB 节点作为硬件测试平台.

TinyOS 操作系统是由一种基于 C 语言的扩展语言 NesC 编写的.通常 NesC 程序可以被 NesC 编译器进行编译,并最终生成嵌入式芯片可执行的文件.以采用 MSP430 的 TelosB 节点为例,图 1 说明了使用 NesC 语言编写的.nc 文件如何生成 MSP430 可执行的.ihex 文件.最终生成的 main.ihex 文件作为一个 TinyOS 镜像(image)包含了整个 TinyOS 操作系统以及开发者编写的应用程序.

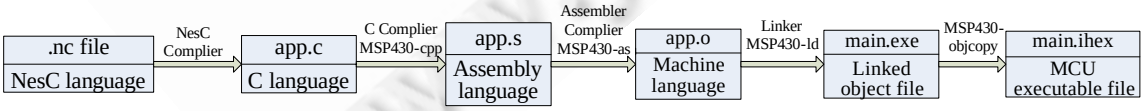

图 1 .ihex 文件生成图

当前大多数传感器节点的嵌入式芯片都采用了 RAM+Flash 的存储结构.本文中的硬件测试平台 TelosB 节点采用了 TI 公司的 16 位单片机 MSP430f1611 作为主控芯片.MSP430f1611 的内部 Flash 与 RAM 统一编址.

默认情况下,.ihex 文件中的全局变量段(包括初始化全局变量.data 段和未初始化全局变量.bss 段)被放入地址空间为 0x1100~0x38FF 的 RAM 中,代码段(.text 段)被放入地址空间为 0x4000~0xFFFF 的内部 Flash 中[14].如果代码段的尺寸超出内部 Flash 的存储范围,则需要将代码放入到外部 Flash 中.

3 EasiLWR 概述

图 2 给出了 EasiLWR 的整体组成框架,它主要由代码转移和函数级代码差异分析两部分组成.函数级代码差异分析包括如何确定差异函数以及如何计算差异代码,并通过仅传输差异代码实现增量式重编程[4−8].代码转移利用属性标记(attribute mark)实现对特定代码的存储位置指定,并且为了保证存储在 RAM 中的代码能够运行,需要利用函数地址重定位和函数的二次调用对新代码的函数地址进行修改.频繁升级的代码通过代码转移被放入 RAM 中执行可以有效地提高更新效率,缩短更新时间.同时,由于避免了对 Flash 的写操作,可以有效延长可重编程时间.

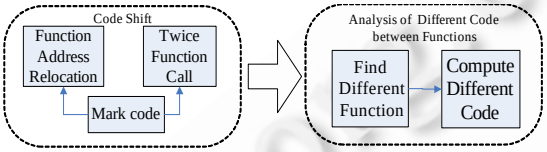

图 2 EasiLWR 结构框图

通常在整个镜像代码中只有部分代码是需要频繁修改的.所以在计算机端,首先通过标记函数的方式将这部分代码预先设定存储在 RAM 中,然后分析新老代码在函数级的差异代码,计算新旧代码的函数级差异代码.随后计算机通过串口将差异代码发往网关节点(sink).网关节点收到差异代码之后,使用无线多跳(multi-hop)方式将差异代码发往待更新节点,如图 3 所示.在节点端,差异代码被下载到 RAM 中进行处理,与存储在 RAM 中的旧代码进行组合后生成新的代码.新代码直接存储在 RAM 中并加以执行,无需再转存到 Flash 中.

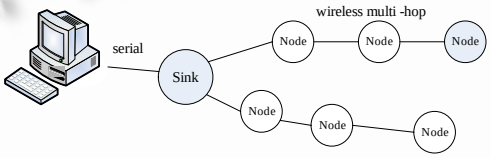

图 3 EasiLWR 传输示意图

4 代码转移

多数无线重编程方法都默认将代码存储在内部 Flash 中并加以执行.但在一个频繁更新代码的场景中[15],连续的重编程意味着需要执行大量的写 Flash 操作,可能导致节点长时间处于编程状态,无法完成正常的工作.另一方面,节点上的大量代码与底层硬件以及操作系统相关,并不需要频繁修改,从而造成了许多更新开销都浪费在无关代码的重建上.EasiLWR 通过标记函数的方法将部分需要频繁升级的代码存储在 RAM 中,并通过修改函数地址确保程序能够被执行.

4.1 标记函数

由于在编译阶段,可以通过链接器 msp430-ld 设置特殊段的起始地址[16],并且 MSP430 的 RAM 与 Flash 统一编址.因此将需要频繁升级的代码段放入特殊段,然后在编译阶段,通过设定特殊段地址将这部分代码存放在 RAM 中.为了将代码放入特殊段,需要对这部分代码段进行标记.特殊段以函数作为基本单位,使用属性(attribute)进行标记.

```
__attribute__((section(".textram"))) static void BlinkC__Boot__booted (void )
__attribute__((section(".textram"))) static void BlinkC__Timer0__fired(void )
```

图 4 标记代码

图 4 中显示了两个被标记的函数,它们被指定存放在.textram 段中,而不是默认的.text 段.另外,如果需要频繁地对若干全局变量进行修改,也可以使用相同的方法,将这些全局变量分别放入.dataram 段(初始化全局变量段)或者.bssram 段(未初始化全局变量段),再设定这两个特殊段的地址到 RAM 中.

由于 RAM 中默认存放了.bss 段和.data 段(存储在 0x1100-0x38ff 的地址中),为了不造成.bssram 段、.dataram 段、.textram 段以及.jmpfun 段(详见第 4.2 节介绍)与默认段发生地址冲突,需要进行两次编译.在第 1 次编译时,设定 4 个特殊段位于相距较远的地址上.进行一次编译后,计算这 6 个段的大小.在确定 4 个特殊段的最终地址后,进行第 2 次编译.

4.2 修改函数地址

当部分代码被存放到 RAM 中之后,有可能出现对存储在 Flash 中的函数引用地址不正确的情况.例如在旧代码中某个与硬件相关的函数的起始地址为 0x4930,存储于 MSP430 的内部 Flash 中.但在新代码中,这个函数的地址为 0x4970.因此,如果新代码中的指令涉及到调用旧代码中存储在 Flash 上的函数,则新代码中对这些函数的引用地址必须根据旧代码的真实函数地址进行修改,即函数地址重定位.

同样,对存放在 RAM 中的函数进行插入或者删除操作时,也会引起函数地址的偏移.为了避免对内部 Flash 中的代码进行修改,需要固定存放在 RAM 中的函数的引用地址.EasiLWR 通过函数的二次调用解决了这个问题:当存储在 Flash 中的代码需要调用存储在 RAM 中的函数时,并不是直接进行调用,而是首先调用跳转函数 jmpfun(被单独放入.jmpfun 段),再由跳转函数 jmpfun 调用存储在 RAM 中的函数.跳转函数存储在固定位置上,当 RAM 中某个函数的位置发生变化时,并不修改 Flash 中对这个函数的引用地址,而只需要修改跳转函数中的引用地址即可.由于跳转函数需要经常更新,因此必须将它存储在 RAM 中执行.

5 函数级代码差异分析

当前的增量式无线编程方法主要通过直接比较可执行文件(.exe),计算差异代码[4,6].这种字节级的差异代码不能保存程序本身的结构信息.本节提出了一种函数级的代码差异比较技术:通过比较包含函数信息的汇编文件(.s)中的函数间差异代码,达到在差异代码中保留程序结构信息的目的.

5.1 差异函数的确定

在计算差异代码之前,首先需要确定哪些函数发生了变化.EasiLWR 采用二级比较机制,即通过分别比较新旧函数的奇偶校检码和 MD4 码的方式确定差异函数.首先对旧代码中的所有函数分别生成奇偶校验码和 MD4 码,并对旧代码本身再生成一次奇偶校检码和 MD4 码.由于生成 MD4 码的计算开销较大,当获得新代码后,并不立即生成新代码的 MD4 码,而是首先生成新代码的奇偶校检码.对新旧代码的奇偶校检码进行比较,如果不同,则说明新代码较旧代码发生了变化,直接开始比较各个函数;如果相同,则计算新代码的 MD4 码,继续与旧代码的 MD4 码进行比较.

新旧函数比较与新旧代码比较类似.首先将新代码中每个函数的奇偶校检码与旧代码中的每个函数的奇偶校检码进行比较,如果不同,则确定了发生改变的函数;然后对所有奇偶校检码相同的函数,分别生成 MD4 码,继续比较,并最终确定所有存在差异的函数.实际上,由于.textram 段中存放了需要频繁升级的函数,通常在比较新旧代码时,如果没有特殊说明,只需要对新旧代码中存放在.textram 段内的函数进行比较.

5.2 计算差异代码

在确定差异函数之后,需要计算这些函数的差异代码.我们使用 Python 语言编写计算差异代码的程序:输入为新旧代码的汇编文件(.s);输出为汇编语言组成的差异代码文件 Diff.s.差异代码文件 Diff.s 的内容如图 5 所示,包括新旧代码的差异代码、差异代码相对于函数第 1 条指令的指令数以及所进行的重编程操作.

其中,/**/中的数字表示相对于函数第 1 条指令的指令偏移数.由于每一条汇编指令对应的机器指令的长度是一定的,所以在得到指令偏移数之后,可以计算得到差异代码相对于函数起始地址的偏移量.Del,Add 和 Res 分别表示 3 种基本重编程操作:删除操作、插入操作和修改操作.修改操作不改变函数的大小,可以直接写入覆

盖旧代码.删除操作可以通过相对跳转(jmp)以及空指令(nop)来实现.所以,当重编程仅存在修改操作和删除操作时,不需要进行代码重建.只有需要进行插入操作时,才进行代码重建.

```
main: call    #__nesc_atomic_start          /*4_Res*/
            mov.b     r15, @r4                    /*5_Del*/
            call   # Scheduler__init /*17_Add*/
            call   # PlatformInit__init /*29_Add*/
Boot__booted:    pop   r4   /*5_Del*/
                    pop   r5   /*6_Res*/
                          …
```

图 5 Diff.s 中的部分差异代码

插入操作或删除操作会改变旧函数的大小,当代码重建以后,紧随其后的所有函数的地址都会发生偏移.一般情况下,对这些函数进行调用的指令都要发生改变,并成为差异代码,由此引起的通信能耗和更新开销将十分巨大.但是由于使用了第 4.2 节中介绍的函数的二次调用,对这些函数的引用地址实际上不会发生改变(跳转函数位置固定,只有跳转函数内的函数引用地址发生改变),从而也不需要修改调用这些函数的指令.另外,需要特别指出的是,由于上述计算差异代码的过程是在计算机上完成的,所以整个计算过程并不会消耗传感器节点的能量,也不会给整个传感器网络增添额外负担.

6 实验结果与分析

为了深入分析 EasiLWR 的性能,本文将从通信开销和更新时间两个方面对 EasiLWR 进行评估.其中通信开销定义为需要无线传输的代码量;更新时间是节点存储代码所需要的总时间.在接下来的实验中,我们将采用 TelosB 节点作为硬件测试平台.

本文设计了 6 种无线重编程场景:

(a) 修改 Blink 中 LED 灯的闪烁频率参数(修改全局变量);

(b) 修改 Blink 的单条指令,使绿色 LED 灯常亮;

(c) 在 Blink 中,插入一行代码到函数 Blink_Timer0_fired 中;

(d) 在 Blink 中,删除 Blink_Boot_booted 函数中的连续 3 行代码,同时删除 Blink_Timer0_fired 函数中的一行代码;

(e) 将 Blink 修改为 CntToLed,CntToLed 通过的 3 个 LED 灯显示收到的计数变量 counter 的最后 3 位值;

(f) 将 CntToLed 修改为 RadioCntToLeds,RadioCntToLed 与 CntToLed 类似,但需要定时发送计数变量 counter 的值.

本文将 EasiLWR 与现有无线重编程方法 Deluge[2],Elon[3]以及 Hermes[6]进行了比较.其中 Deluge 是标准 TinyOS 无线重编程方法;Elon 使用 RAM 存储频繁升级的代码;Hermes 采用了基于比特级的差异对比方法,有效降低了传输代码量.实验结果表明,EasiLWR 无论是在通信开销还是在更新时间方面都具有较明显的优势.

6.1 通信开销

表 1 显示了在场景(a)~场景(f)中,Deluge,Elon 以及 EasiLWR 所需要传输的代码量.观察发现,Deluge 的通信开销十分巨大.这是由于 Deluge 需要传输整个 TinyOS 镜像(image)以及重编程协议造成的.以场景(a)为例,当使用 Deluge 重编程时,传输的 Blink 镜像代码本身仅占总传输代码的 11.5%,而真正要修改的代码只占总传输代码的 0.043%.Elon 通过设定全局变量为可替换(replaceable),可以只对全局变量进行修改.但是由于没有采用代码的差异比较,当新旧代码的某个函数不同时,需要传输整个存在差异的函数.EasiLWR 提出了函数级的代码差异比较技术,有效降低了通信开销.在场景(a)中只需要传输存在差异的全局变量,Elon 的通信开销与 EasiLWR 持平,而在场景(b)~场景(d)中,Elon 的通信开销较多地浪费在了传输函数中的无关代码上.在场景(e)和场景(f)中,若干函数被分别被插入到旧代码中.新旧代码的差异是整个函数,因此,EasiLWR 的函数级的代码差异对比没有起作用,与 Elon 传输代码量相同.

表 1 无线重编程所需传输代码量(bytes)

	场景(a)	场景(b)	场景(c)	场景(d)	场景(e)	场景(f)
Deluge	23 110	23 110	23 114	23 096	32 008	32 198
Elon	8	58	62	96	86	278
EasiLWR	8	14	14	16	86	278

6.2 更新时间

表 2 分别显示了在场景(a)~场景(f)中,Hermes,Elon 以及 EasiLWR 的更新时间.Hermes 与 EasiLWR 类似,采用了差异比较技术,传输代码量与 EasiLWR 相同.在场景(a)中,3 种方法都可以独立地对存储在 RAM 中的全局变量进行修改,更新时间基本相同.但是,如果需要修改代码,Hermes 会首先将差异代码存入外部 Flash 进行重建,然后将重建后的新代码重新读入 MSP430 内部 Flash,这样做,导致更新效率较低,更新时间长.Elon 则使用 RAM 存储并执行需要频繁升级的代码,大幅降低了更新时间.EasiLWR 在分析 3 种基本重编程操作的基础上,可以在不重建代码的前提下直接执行差异代码,进一步缩短了更新时间.以场景(b)和场景(d)为例,当 EasiLWR 执行修改操作或删除操作时,不需要重建代码,更新时间远小于 Elon.在场景(c)、场景(e)、场景(f)中,必须进行插入操作,EasiLWR 也需要重建代码,但是由于只需要重建存储在 RAM 中的代码,更新时间与 Elon 相近.

表 2 无线重编程所需更新时间(ms)

	场景(a)	场景(b)	场景(c)	场景(d)	场景(e)	场景(f)
Hermes	0.011 7	3 786.1	3 451.9	3 552.6	3 200.8	3 219
Elon	0.015 2	1.26	1.17	1.32	1.25	1.37
EasiLWR	0.013 1	0.027	1.39	0.024 3	1.31	1.27

7 结 论

通信开销大和更新时间长是制约当前无线重编程应用的两个重要问题.本文针对如何解决这两个问题展开了讨论,并提了一种新的无线重编程方法 EasiLWR.EasiLWR 通过分析函数级代码差异,计算新旧代码的差异代码,并通过传输差异代码,减少了通信开销.为了缩短更新时间,EasiLWR 通过代码转移实现将部分代码存储在 RAM 中执行,尽量避免对内部 Flash 的写操作.同时考虑到电池电压过低时,写 Flash 操作可能发生无法预期的错误,所以尽量避免写 Flash 操作也能延长可重编程时间.实验结果表明,EasiLWR 较现有无线重编程方法在节约能耗方面有较大提高.在未来的工作中,我们将把 EasiLWR 应用到更多的无线传感器网络操作系统以及传感器节点上,并进一步测试其性能.

References:

[1] MSP430x1xx Family User's Guide (Rev. F). Available: http://focus.ti.com/lit/ug/slau049f/slau049f.pdf

[2] Hui JW, David C. The dynamic behavior of a data dissemination protocol for network programming at scale. In: Stankovic JA, ed. Proc. of the ACM 2nd Int'l Conf. on Embedded Networked Sensor Systems (SenSys 2004). Baltimore: ACM Press, 2004. 84–91. [doi:10.1145/1031495.1031506]

[3] Dong W, Liu YH, Wu X, Gu L, Chen C. Elon: Enabling efficient and long-term reprogramming for wireless sensor networks. In: Misra V, ed. Proc. of the ACM Int'l Conf. on Measurement and Modeling of Computer Systems (SIGMETRICS 2010). New York: ACM Press, 2010. 49–60. [doi:10.1145/1811099.1811046]

[4] Panta RK, Bagchi S, Midkiff SP. Zephyr: Efficient incremental reprogramming of sensor nodes using function call indirections and difference computation. In: Proc. of the 2009 Conf. on USENIX Annual Technical Conf. California: USENIX Association Press, 2009. 32–46.

[5] Koshy J, Pandey R. Remote incremental linking for energy-efficient reprogramming of sensor networks. In: Cayirci E, ed. Proc. of the 2nd European Workshop on Wireless Sensor Networks (EWSN 2005). Istanbul: IEEE Press, 2005. 354–365. [doi: 10.1109/EWSN.2005.1462027]

[6] Panta RK, Bagchi S. Hermes: Fast and energy efficient incremental code updates for wireless sensor networks. In: Kurose J, ed. Proc. of the 28th Int'l Conf. Computer Communications (INFOCOM 2009). Rio de Janeiro: IEEE Computer Society, 2009. 639–647. [doi: 10.1109/INFCOM.2009.5061971]

[7] Hu J, Xue CJ, He Y. Reprogramming with minimal transferred data on wireless sensor network. In: Ni ML, ed. Proc. of the 6th IEEE Int'l Conf. on Mobile Ad-hoc and Sensor Systems (MASS 2009). Macau: IEEE Press, 2009. 160−167. [doi: 10.1109/MOBHOC.2009.5337000]

[8] Jeong J, Culler D. Incremental network programming for wireless sensors. In: Znati T, ed. Proc. of the 1st IEEE Communications Society Conf. on Sensor and Ad Hoc Communications and Networks (SECON 2004). Santa Clara: IEEE Press, 2004. 25−33. [doi: 10.1109/SAHCN.2004.1381899]

[9] Han CC, Kumar R, Shea R, Kohler E, Srivastava M. SOS: A dynamic operating system for sensor networks. In: Shin KG, ed. Proc. of the ACM 3rd Int'l Conf. on Mobile Systems, Applications, and Services (MobiSys 2005). Seattle: ACM Press, 2005. 163−176. [doi: 10.1145/1067170.1067188]

[10] Dukels A, Gronvall B, Voig T. Contiki—A lightweight and flexible operating system for tiny networked sensors. In: Proc. of the 29th Annual IEEE Int'l Conf. on Local Computer Networks (LCN 2004). Bonn: IEEE Computer Society, 2004. 455−462. [doi: 10.1109/LCN.2004.38]

[11] Levis P, Culler D. Mat´e: A tiny virtual machine for sensor networks. In: Gharachorloo K, ed. Proc. of the ACM 10th Int'l Conf. on Architectural Support for Programming Languages and Operating Systems (SIGOPS 2002). San Jose: ACM Press, 2002. 85−95. [doi: 10.1145/635508.605407]

[12] Levis P, Gay D, Culler D. Active sensor networks. In: Vahdat A, ed. Proc. of the USENIX/ACM 2nd Conf. on Symp. on Networked Systems Design & Implementation (NSDI 2005). Boston: USENIX Press, 2005. 343−356.

[13] TinyOS. http://www.tinyos.net

[14] MSP430F15x, MSP430F16x, MSP430F161x Mixed Signal Microcontroller. http://focus.ti.com/lit/ds/symlink/msp430f1611.pdf

[15] Sun NH, Xu ZW, Li GJ. Sea computing: A novel computing model of internet of things. Communication of China Computer Federation, 2010,(2):39−43 (in Chinese with English abstract).

[16] GNU Binutils: Loader and Linker. http://sourceware.org/binutils/docs-2.21/ld/index.html

附中文参考文献:

[15] 孙凝晖,徐志伟,李国杰.海计算:物联网的新型计算模型.中国计算机学会通讯,2010,(2):39−43.

邱杰凡(1984－),男,河南新乡人,博士生,主要研究领域为无线传感器网络,分布式计算.

石海龙(1986－),男,博士生,主要研究领域为无线传感器网络,嵌入式系统.

李栋(1979－),男,博士,助理研究员,主要研究领域为无线传感器网络,无线自组织网络.

崔莉(1962－),女,博士,研究员,博士生导师,主要研究领域为传感器技术,无线传感器网络.