面向软件定义架构的无线传感器网络

董 玮 陈共龙 曹晨红 罗路遥 高 艺
(浙江大学计算机科学与技术学院 杭州 310027)

摘 要 近年来,嵌入式技术与无线网络技术深度结合,催生了可计算 RFID、嵌入式传感网等新兴领域.这些系统由大量廉价的节点组成,应用前景广泛.在传统设计中,这些系统通常是根据应用定制的.根据应用定制的系统具有开发简便、运行高效等优点,但不适合未来大规模部署.这是因为如果这些系统跟应用密切绑定、难以更新,那么系统一经部署就难以更新其软件,从而阻碍了软件创新的进程.软件定义的思想可以有效解决该问题.当前,软件定义网络成为计算机网络中一个热门的研究领域.传感器网络的软件设计与因特网的软件设计存在诸多差异,其最大的差异在于,传感器网络主要以信息的采集为核心,而因特网主要以信息的传输为核心.此外,传感器节点还具有体积小、电池续航能力有限、价格低廉等特点.文中主要调研了设计软件定义传感器网络(Software-Defined Sensor Networks,SDSNs)架构的相关工作,列举了在设计一个通用、高效的软件定义传感器网络架构时可能遇到的挑战,并回顾了一些有用的技术.这些技术有的来自于现有方案,有的能够直接被用来解决一部分挑战.此外,文中还从软件定义功能的角度,进一步地对目前通用、高效的软件定义传感器网络架构及其采用的技术进行了分类.我们认为,软件定义传感器网络架构将在已部署的网络中起到至关重要的作用,并带来一场新的技术变革.

关键词 软件定义网络;无线传感器网络;重编程;网络架构;软件定义测量;物联网;信息物理融会系统
中图法分类号 TP393 **DOI 号** 10.11897/SP.J.1016.2017.01779

Towards a Software-Defined Architecture for Wireless Sensor Networks

DONG Wei CHEN Gong-Long CAO Chen-Hong LUO Lu-Yao GAO Yi
(*College of Computer Science and Technology, Zhejiang University, Hangzhou* 310027)

Abstract Recent years have witnessed the deep integration of embedded systems and wireless networked systems such as computational RFIDs and embedded sensor networks. These systems comprise of tiny embedded chips, low-power radios, and various sensors. They allow for an instrumentation of the physical world at an unprecedented scale and density, thus enabling a new generation of monitoring and control applications. Traditionally, these systems are highly application customized because of their resource constraints. Customized systems have the benefits of easy developments and good performance. They are, however, not a good fit for wide deployments in the future. If these systems were highly application dependent and inflexible to upgrade, our technical innovations will be severely hindered since the replacement of a large number of sensor products take a long time. This situation would be similar to today's Internet where the deployment of a new protocol is not easy. This calls for a software defined architecture. Software-defined network has been a hot research topic for the networking community. Software design for sensor networks has many differences from the Internet. Most importantly, sensor networks are primarily used for information gathering while the Internet is primarily used for

收稿日期:2016-08-24;在线出版日期:2017-03-10.本课题得到国家自然科学基金(61472360,61502417)、浙江省重点研发计划(2017C02044)、CCF-腾迅犀牛鸟科研基金、中央高校基本科研业务费专项资金资助.**董 玮**,男,1982 年生,博士,教授,中国计算机学会(CCF)高级会员,主要研究领域为物联网、传感器网络测量与移动计算.E-mail:dongw@zju.edu.cn.**陈共龙**,男,1992 年生,博士研究生,主要研究方向为无线网络与移动计算.**曹晨红**,女,1991 年生,博士研究生,主要研究方向为物联网、传感器网络测量.**罗路遥**,男,1992 年生,硕士研究生,主要研究方向为物联网、传感器网络测量.**高 艺**,男,1987 年生,博士,助理研究员,主要研究方向为传感器网络协议设计、网络测量算法、无线与移动计算.

information transportation. Moreover, sensors are characterized as resource constrained due to the small form factor, limited battery supply, and low prices. In this article, we focus on the design of software-defined architecture for wireless sensor networks. The technology of traditional software-defined network mainly focuses on how to make the routing behavior software-defined. Software-defined sensor networks focus on not only software-defined routing, but also other aspects such as software-defined sensing. For the design of a generic and efficient software-defined sensor network architecture, we first examine some major challenges, e. g. , the design of a generic software-defined architecture, efficient allocation of resources, rapid update and real-time control, maintenance of network consistency, etc. Among these challenges, the design of a generic software-defined architecture for sensor networks is especially important, and it is the basis for controlling the network in a flexible way. The efficient allocation of resources, rapid update and real-time control, and maintenance of network consistency are also key challenges in implementing software-defined sensor network. The challenge of security already exists in traditional sensor networks as well as the Internet. However, how to design a secure and lightweight security mechanism in software-defined sensor network faces new challenges. To address part of the above challenges, we review some useful techniques. Then, we also present a novel taxonomy for software-defined sensor network according to different abstractions of functionalities, including software-defined sensing, software-defined routing, software-defined measurements, software-defined debugging, and others. The purpose of this article is to summarize the current designs in software-defined sensor network, so as to explore a larger research space in this direction. In the end, we also summarize some possible future directions, e. g. , the possible combination of wireless reprogramming and remote debugging, and the design of agile programming model. In the future, we believe that a well-designed software-defined sensor network architecture can greatly facilitate software development for deployed networks, allowing rapid technical innovations.

Keywords software-defined network; wireless sensor networks; reprogramming; network architecture; software-defined measurement; Internet of Things; Cyber-Physical System

1 引 言

近年来,基于嵌入式系统与无线网络系统深度融合的技术,受到了广泛的关注,这些技术包括可计算射频标签(Computational RFID)、传感器网络(Wireless Sensor Networks,WSNs)等.这些由微型嵌入式芯片、低功耗无线设备和各种传感器组成的系统,正以空前的规模和密度部署在我们的生活中,催生了对新一代监测、控制软件的需求.由于资源的限制,这些系统一般被用在特定的应用中.正如早期对传感器网络的定义那样:

"传感器网络是面向特定应用的.传统的网络在设计初期就考虑到了在不同应用中的通用性,但我们有理由相信传感器网络更加适用于某些特定的场景.比如,传感数据的融合、缓存或者请求转发.这与传统网络中的点对点通信有着显著的不同[1]."

面向特定应用的系统虽然开发代价低且性能良好,但它们并不适合在将来广泛的部署.试想在不远的未来,我们的周围遍布着大量的智能传感器,但它们只能在特定的应用中使用.这时,一旦软件需要升级,我们将要花费大量的时间去更新数量庞大的智能传感器,这无疑对技术的革新造成了巨大的阻碍.无独有偶,当前的因特网也面临着类似的困境.因此,软件定义的概念受到了广泛的关注.基于这一架构,软件的设计将更加灵活,应用的升级将更加简单.

在因特网领域,软件定义网络(Software-Defined Networks)已经是一个受关注的话题.Gartner 公司把软件定义一切(Software-Defined Everything)列为 2015 年十大技术趋势之一①.软件定义网络将数据层与控制层分离,路由器与交换机只需要依据控

① Gartner Identifies the Top 10 Strategic Technology Trends for 2015. http://www.gartner.com/newsroom/id/2867917

制层的规则进行数据包的转发，使整个网络变得更加灵活可控. OpenFlow[2] 是应用最广泛的软件定义网络协议，其提供了一系列的规范. Liu 等人[3] 对软件定义网络及 OpenFlow 中的重要概念、应用场景、语言抽象、安全问题、在光纤网络中的实现等都做了归纳及阐述. 随着传感器网络变成一个越来越重要的平台，软件定义传感器网络也成为了研究热点[4].

软件定义传感器网络是指运用了软件定义技术进行感知、路由、测量等任务的新型传感器网络. 软件定义传感器网络和传统软件定义网络既有联系也有区别. 传统的软件定义网络技术关注信息的传输，主要研究软件定义路由. 由于传感器网络还需进行数据感知，软件定义传感器网络除了关注软件定义路由以外，同时也要关注软件定义感知等其他方面. 和传统传感器网络相比，我们总结的软件定义传感器网络优势如下：

(1) 资源的充分利用. 一个公共的传感器平台可被用于不同的应用场景中.

(2) 简易的配置和管理. 系统管理员可以在网络部署后动态地配置系统.

(3) 快速的更新. 从长远来看，一个好的软件定义传感器网络构架可以让系统快速的更新以满足未来的需求.

(4) 快速的技术革新. 软件定义传感器网络能够加速软件的部署和测试，从而加速技术的革新.

本文调研了现有的软件定义传感器网络架构的设计. 传感器网络与因特网的软件设计存在诸多差异，其最大的差异在于传感器网络主要以信息的采集为核心，而因特网主要以信息的传输为核心. 此外，传感器节点还具有体积小、电池续航能力有限、价格低廉等特点. 本文列举了在设计一个通用、高效的软件定义传感器网络架构时可能遇到的挑战，并回顾了一些关键的技术. 这些技术有的来自于现有方案，有的能够直接被用来解决一部分挑战. 此外，本文还从软件定义功能的角度，进一步地对目前通用、高效的软件定义传感器网络架构及其采用的技术进行了分类.

2　软件定义传感器网络架构

最近几年，研究者提出了多种有关软件定义传感器网络的方案. Zeng 等人[5] 提出了“感知即服务”(Sensing-as-a-Service)架构. 在这种架构中，系统管理员可以基于无线重编程技术，按需对传感器网络中部分节点进行重编程，部署指定的应用. 在其他几种方案中[6-9]，软件定义传感器网络的架构与软件定义因特网的架构是类似的. 如图 1 所示的软件定义传感器网络架构，最上层为应用层；中间层为逻辑控制层，主要根据类 OpenFlow 协议与下层节点通信、分发指令；下层为包含传感器节点的数据转发层，每个传感器节点都有一个类 OpenFlow 流表(Flow Table)的抽象，它们根据逻辑控制层发来的指令配置流表中的行为(Action)，根据收到的数据包匹配流表中对应的行为，从而执行相应的操作.

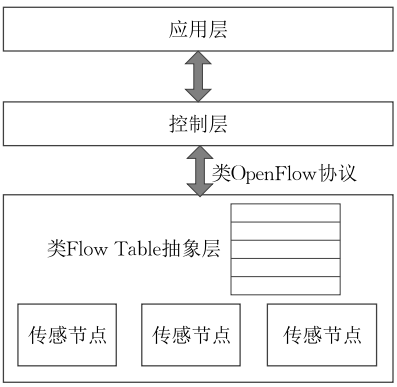

图 1　软件定义传感器网络架构

在中间层的逻辑控制层中，控制器将与各传感器节点建立连接，并传输控制指令. 在传统的 OpenFlow 协议中，控制器与交换机之间使用 TCP/IP 协议建立连接. 这种做法对于不使用 IP 地址的传感器网络来说并不适用. 在传统的传感器网络研究中，已经有许多关于传输协议设计方面的研究①. Gante 等人[10] 认为，软件定义网络中的逻辑控制层是软件定义网络的精华所在. 因为逻辑控制层拥有全局网络的整体视图，从而能让网络管理员做出更优的传感器网络资源管理与配置. 因此，Gante 等人设计了一种新型的基于软件定义传感器网络范式的基站架构，以解决传统传感器网络中的资源管理问题(如传感器节点的移动、定位等).

在下层的数据转发层中，数据包以数据流为单位进行处理. 一个数据流是指用户定义的一组具有相同性质的数据包，这些性质由流表中的匹配规则来指定. 例如，流表中某一项的匹配规则为“源节点 ZigBee 地址为 0x796F”，匹配该规则的所有数据包被当作同一个数据流，并被赋予相同的行为. 该行为也在同一条流表项定义，例如，“丢弃该包”.

基于软件定义传感器网络的架构可以在同构网

① A compiled list of transport protocols for wireless sensor networks. http://www1.i2r.a-star.edu.sg/~luot/wsn-tp.pdf, 2012, 2

络中实现.在同构网络中,一般有两种节点:具有转发功能的普通节点和具有向普通节点发出执行命令的控制节点.开发者可以依据 API 文档实现与控制节点的交互.

GreenOrbs[11](绿野千传)是一种典型的同构传感器网络,该网络包含数百个传感器节点以及一个公共汇聚点.在此类场景中,汇聚节点一般是控制节点,而其他节点一般是转发节点.大部分传感器网络是以多跳的方式通信的,在规模较小的传感器网络中,一般利用单一汇聚节点向其他节点发送控制命令的方式搜集数据.但对于大规模传感器网络而言,此类部署方案通常会导致较大的通信开销.

为了解决这个问题,可以采用多个控制节点分布式部署的方案.如图 2 所示的多个控制节点的软件定义传感器网络架构中,黑色节点代表控制节点,白色节点代表普通节点.每个普通节点选择最近的控制节点通信,从而减少逻辑控制层的通信开销.在这样的架构中,每个控制节点都有各自的网络逻辑视图,为了使不同的控制节点能够协同工作,维持这些控制节点中网络逻辑视图的一致性是十分重要的.

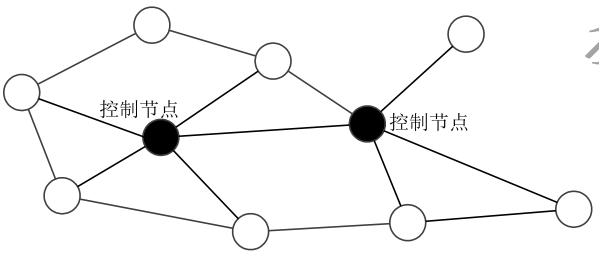

图 2 多个控制节点的软件定义传感器网络架构
(黑色节点代表控制节点,白色节点代表普通节点)

由于控制节点要执行比普通节点更复杂的功能(例如,存储网络拓扑图、响应来自普通节点的请求等),软件定义传感器网络可以使用计算能力更强的节点作为控制节点,称为"主节点"[12].在这样一个由异构节点组成的分层架构中,主节点层可以实现多节点的数据融合和多节点的应用逻辑等功能,既简化了应用开发流程,又提高了软件复用率.

根据网络本身架构不同,软件定义传感器网络的部署可以采取不同的方案.接下来将给出软件定义传感器网络进行实际应用时所存在的挑战以及关键技术.

3 挑战及关键技术

现在已经有许多软件定义传感器网络架构的设计,但大都停留于原型程序,或规模有限的测试系统.一般来说,高效与通用二者不可兼得.例如,近几年来吸引了众多研究者注意的重编程技术,可以对网络中节点的程序进行代码级更新,达到很好的通用性.但由于它巨大的开销和不可靠性,很难直接应用于现有的网络中.另一方面,为了减少软件定义传感器网络的开销,许多方案仅将目标限定在软件定义路由,忽视了软件定义感知、软件定义测量等其他方面的功能,因此通用性不佳.

如表 1 所示,设计一个通用、高效的软件定义传感器网络,面临以下 5 项关键挑战:通用的软件定义架构、资源的有效分配、快速更新与实时控制、一致性以及安全性.其中,面向传感器网络设计一个通用的软件定义架构尤为重要,它是基于软件定义的思想创建上层应用的基础.资源的有效分配、快速更新与实时控制以及一致性是具体实现软件定义传感器网络时面临的三项关键挑战.安全性的挑战在传统的传感器网络和软件定义网络中就已存在,然而,如何在软件定义的传感器网络中设计一个安全、轻量的系统仍然是一个开放性的问题.现有工作分别对这些挑战有一定程度上的解决,然而在某些场景中并不适用.本节系统总结了一些关键技术以应对不同的挑战.

表 1 挑战及关键技术

挑战	关键技术
通用的软件定义架构	系统抽象、虚拟化、动态链接和加载、动态代码注入
资源的有效分配	能量最小化、混合整数线性规划
快速更新与实时控制	连通支配集计算、有利干涉、流水线传输机制
一致性	两阶段更新
安全性	Merkle 散列树、可信基随机阈值限速检测算法、最大熵检测算法

3.1 通用的软件定义架构

挑战

现在已有许多方案实现了不同的软件定义功能.那么,如何设计一个通用的、开销最小的架构,以支持不同的软件定义功能?

关键技术

系统抽象技术是软件定义架构中的关键技术,现有方案已经实现了不同层次的抽象.软件定义路由实现了逻辑控制层与普通节点层之间的抽象.普通节点中的程序根据流表中的条目,执行不同的路由任务,网络管理员通过修改流表的参数来动态的配置不同的路由策略.例如表 2 所示的一个简单流表,在软件定义传感网络中作为数据转发的依据.当

普通节点收到的数据包满足匹配规则(Matching Rule)时传感器节点将对其执行特定的行为(Action).在匹配规则中,字段 Operator(Op.)指定关系操作符,字段 Offset 与 Size 指定要匹配的数据起始字节及大小,字段 Val.为要满足的值;在行为中,类型字段(Type)指定对满足条件的数据包要执行的操作类型.例如第一条规则表示,如果数据包起始节点(第 2～3 字节的值)为 A,则将其转发到节点 D.

表 2　软件定义传感器网络流表示例

匹配规则				行为		
Op.	Size	Offset	Val.	Type	Offset.	Val.
=	2	2	A	Forward	0	D

DT(Declarative Tracepoints)[13]和 Dylog[14]实现了对运行在普通节点上进行系统调试的抽象,它们将特殊的程序(跟踪点或日志记录语句)注入到原始程序中,以动态地执行预设的任务.此外,如果将整个系统(包括数据段和代码段)抽象成模块,利用动态装载器更新该模块,就实现了整个系统层次的动态更新.

为了实现以上所述的抽象技术,可以使用以下的方法:

(1)虚拟化技术.虚拟化技术是一种用于改变传感器网络功能的常用方法.通常来说,在虚拟机上运行的程序代码比在物理机上运行的程序代码(本地代码)要更为紧凑.因此,虚拟机通常应用于有频繁重编程需求的传感器网络.大多数传感器网络的虚拟机为了适应不同的应用程序而被设计成了高度可配置的系统.

虚拟机可以设计成通用的,如 Magnet OS[15]、Contiki OS① 等.为了能在资源受限的传感器上运行,Java 类通常存储在闪存芯片的只读存储器(ROM)上,而不是随机访问内存(RAM)上.Java 程序可以通过声明原生(Native)Java 函数来调用原生代码,从而提高运行时的效率.

虚拟机也可以设计成面向特定应用的.比如,Maté[16]是一个运行在 TinyOS[17]系统上基于虚拟机架构的中间件层.一般来说,Maté 将代码划分为 24 个指令大小的块,对于更大的程序,Maté 将其划分为以子程序为单位的块(子程序个数限制为 4).进程间的通信使用类似于 TinyOS 的消息传递机制.目前,Maté 目标语言仅支持汇编语言或字节码.

(2)动态链接和加载技术.模块化操作系统为代码的执行或加载提供了动态链接器或加载器.一个可加载的模块包含了程序的代码,程序代码中通常包含指向系统函数或变量的引用.在执行前,这些引用必须事先修正为函数或变量的物理地址,这个过程称为链接.

SOS[18]、Contiki OS① 和 FlexCup[19]都提供了对动态加载技术的支持,在这些系统中,独立的模块可以被动态加载到传感器节点上.例如,在 Contiki OS 中,应用程序使用预链接的二进制模块来进行动态加载.在编译 Contiki OS 系统内核时,编译器将生成一个映射文件,包含系统内核中所有全局可见的函数和变量与其地址之间的映射.之后,应用程序利用该映射预链接 Contiki OS 模块.模块中的机器码不仅包含指向系统函数或变量的引用,而且包含指向模块内函数或变量的引用.函数的物理地址将根据加载时的内存地址发生变化.因此,引用的地址必须更新为模块加载时函数或变量的物理地址,更新这些引用的过程称为重定位.

当对一个模块进行链接和重定位时,程序加载器从代码执行的地方将链接及重定位后的本地代码复制到内存中.具体来说,可加载模块的标准格式是为 32 位或 64 位的 PC 或工作站设计的(例如 ELF(Executable and Linking Format)文件格式),对于 8 位或 16 位的目标传感系统,ELF 格式中字段的高 16 位未被使用,增加了不必要的开销.ELF 文件不仅包括程序代码和数据,还包括其他附加信息,如符号表(Symbol Table)、所有外部未解析符号名以及重定向表.同时,链接和重定位过程通常非常复杂,会增加模块加载时的开销.因此,研究者们提出了多种优化方法,进一步减小 ELF 文件大小的方法,提高代码加载效率.

Dunkels 等人[20]设计了 CELF 格式(Compact ELF)来替代 ELF 格式.CELF 文件包含了与 ELF 文件相同的内容,但用 8 位或 16 位的数据类型来表示,大大减少了文件大小,通常是相应 ELF 文件大小的一半.

(3)动态代码注入技术.动态代码注入允许用户在程序运行时将代码注入到原始程序中.动态代码不是通过加载来执行,而是通过特殊的方法,如 Trampoline[21].其中,应特别关注以下问题:如何引导程序执行注入的代码、如何处理指令覆盖、如何在注入代码执行完后将控制权交回给原始程序.

Trampoline[21]是一种典型的动态代码注入技术.其中,Trampoline 指的是一小段用于启动其他

① The Contiki OS: The Operating System for the Internet of Things. http://www.contikios.org, 2011

程序的代码. 当 Trampoline 被注入到目标程序中后,源程序部分指令被覆盖,同时目标程序会被 Trampoline 重定向到一段补丁(Patch)程序中执行,执行完成后,接着执行被覆盖的指令并返回原始程序.

总结

在软件定义传感器网络中设计一个通用、低开销的架构的核心问题,在于对软件定义功能的抽象及实现. 现有的对于软件定义传感器网络的研究集中在软件定义路由上,通过借鉴软件定义网络中的流表实现了对路由任务的抽象. 而对于软件定义感知、软件定义测量等其他功能的实现,我们可以从传感器网络中的其它通用技术中获得启示,将功能程序抽象为模块,通过动态链接和加载,动态代码注入等技术,在系统运行时,动态修改传感器网络的功能. 然而动态链接和加载通常带来很大的通信开销与存储开销,动态代码注入需要特殊组件的支持,如何运用和改进这些技术以满足传感器节点低开销的需求,值得更进一步的研究.

3.2 资源的有效分配

挑战

借助软件定义的架构,用户可以根据上层应用的需求(如感知的质量、能耗的均衡等)动态地分配网络资源. 那么,如何分配网络资源,才能更好地满足用户需求呢?

关键技术

在一个优化资源分配的问题中,目标函数通常是一个最小化开销函数,或者是一个最大化收益函数. 例如,Zeng 等人[22]在保证了感知质量的情况下,考虑了软件定义传感器网络中的能量最小化问题. 一个给定的感知任务通常需要多个节点达到某一特定的感知质量(如覆盖率). Zeng 等人考虑了三种与服务质量(Quality of Service,QoS)相关的问题:传感器的激活,任务的映射,以及感知程序的调度. 此外,在利用软件定义架构进行灵活设计的同时,还应当把实现软件定义架构时所造成的能量损耗,资源占用计算在内,以保证网络的正常运行.

Zeng 等人[22]将该感知质量问题形式化成了一个混合整数二次规划(Mixed Integer Quadratic Program,MIQP)问题,该问题可以通过线性化转化为一个有着较低计算复杂度的混合整数线性规划(Mixed Integer Linear Program,MILP)问题. 其中,传感器网络覆盖率、可调度性、存储空间等都做为该问题的约束条件. 通过求解这个问题,Zeng 等人能够得到一个满足用户需求的资源分配方案. 该方案在多任务的软件定义传感器网络中,在同时考虑了网络可调度性和质量保证的条件下,是第一个对能量最小化问题的研究. 该研究还针对两类网络动态性的问题提出了在线算法. 一类是由于网络操作,部署新应用时产生的动态性;另一类是网络节点的变化产生的动态性,例如能量耗尽导致的节点移除,或是新的节点加入.

传感器网络中的资源、能耗、计算能力、数据冗余性、应用本身等之间具有很强的相关性. 在传感器网络的负载均衡路由算法[23-24]中,每条链路的能量消耗被建模成链路的权重. 通过最小化路由链路的总权重,就能构建出最优的路由树. 然而,在软件定义传感器网络中,随着应用的更新,不同链路的能量消耗是动态变化的,传统的负载均衡路由算法因而不能求解出最优树,限制了在环境动态变化中的适应性.

Huang 等人[25]提出了基于强化学习机制的冗余数据过滤技术和负载均衡路由技术,以提高能量的利用率以及对环境变化的适应能力. 计算复杂性高的数据融合和路由管理集中在控制层,数据转发层仅处理计算复杂性低的算法. 控制层运行着强化学习机制. 节点与应用之间的交互(惩罚或奖励)事件被用来更新强化学习的参数,从而提高系统在复杂动态环境中的节能机制的自适应性. 此外,基于时间序列预测的自回归滑动平均模型,能够挖掘传输数据流中表现出周期性的冗余数据,从而为冗余过滤提供可能. 基于强化学习机制的冗余数据过滤技术和能够适应环境变化的负载均衡路由技术让传感器网络中的资源得到了有效的利用. 在该项研究中,作者实现了系统原型,并通过实验结果表明在保障 QoS 的情况下,该原型降低了能耗,提高了网络的自适应性.

由于传感器节点中有限的电池容量,节能始终是传感器网络面临的重要问题之一. 传感器网络中变化频繁的拓扑和更新频繁的路由表导致传感器网络的能量消耗明显. 传统的路由协议设计方案也试图针对拓扑变化频繁的情况尽可能减少能源消耗. 而在软件定义传感器网络中,数据层与控制层分离,使得数据层的传感器节点只保留转发的任务,不需要路由决策,从而减少了能耗[26].

总结

传感器节点的资源有限,在追求能量最小化的同时,还应满足用户对于数据感知的需求. Zeng 等人[22]的工作是第一个针对软件定义传感器网络的,在保证传感器节点的可调度性和感知数据质量的前

提下，对能量最小化问题的研究. 基于软件定义传感器网络中控制层和数据层分离的特点，Huang 等人[25]通过将数据层的复杂计算操作转移到计算能力更强、资源更丰富的控制层，对冗余数据进行过滤等方式，也能大幅减少数据层中普通传感器节点能量的消耗. 然而，部署过多计算能力更强、资源更丰富的控制层节点会产生较大的资金开支，且会引入较大的通信开销. 因此，如何在给定的传感器网络拓扑下，在适当的位置配置适当数量的控制层节点，以最小化资金开支与通信开销，是一个值得未来深入研究的问题.

3.3　快速更新与实时控制

挑战

由于资源的限制，控制消息和数据消息的传输使用了相同的信道，这就可能引发冲突、拥塞或者干扰. 因此，如何实现实时、可靠的带宽内控制也是一个亟待解决的问题.

关键技术

正如前文所述，在一个典型的软件定义传感器网络中，有两种类型的节点：控制节点和普通节点. 拓扑构建的目标是设计一个部署控制节点的方案，在保证控制节点和普通节点的通信的情况下最小化部署成本.

现有的拓扑算法已经能有效的解决此问题. 比如，在连通支配集(Connected Dominating Set，CDS)规划中，需要找到一个最小的节点集合(控制节点)以满足：连通支配集是互连的；非连通支配集的节点直接连向连通支配集节点. 因为控制节点和普通节点之间的距离是有边界的，所以可以更容易的获得快速可靠的控制.

还有一些方案使用分布式算法选择连通支配集节点. Cheng 等人[27]提出了一种单领导节点的算法，初始时所有节点都打上白色、非激活的标签. 算法开始时，领导节点首先将自己标记成黑色，成为支配节点. 由领导节点开始，其他节点按如下规则进行：如果一个白色节点的某一邻居节点是支配节点，该白色节点将自己标记成灰色节点，即成为被支配节点；如果一个非激活的白色节点的某一邻居节点是被支配节点，该白色节点将自己转变成激活状态，颜色仍然保持白色. 拥有最大的实际度数(白色邻居的数量)的激活节点竞争成为支配节点，同时，为了保证连通性，竞争成功的节点的灰色父节点标记成黑色，成为支配节点. 最后，所有的黑色叶子节点转变成灰色，成为被支配节点. 当所有节点都标记成黑色或者灰色，且所有黑色节点能够构成连通支配集时，算法停止. 该方法解决了分布式传感器网络中选择连通支配集节点的问题.

在拓扑构建的同时，还需要保证控制器与传感节点间的通信必须是快速和可靠的. 传统方法通过使用三次握手机制来实现可靠性. 接收到最新数据的节点发送广播表明其已收到的数据. 其他节点如果不具有该新数据，则对广播节点进行请求. 然后广播节点将根据收到的请求发送新数据到其他节点. 最终，所有的网络节点将收到新数据.

这种方法是可靠的，但并不快速. 近年来，有很多研究工作致力于低功率无线电的有利干扰(Constructive Interference，CI). 在传统的无线通信中，多个发送者同时发送数据会造成互相干扰，导致接收者无法正确解码. 而在 CI 技术中，多个发送者几乎同时(时间间隔小于 0.5 μs)发送(或转发)相同内容的数据. 在这种情况下，多个发送者发送的数据在接收者处相互叠加，不但没有造成干扰，反而增强了被正确接收的概率，这也是该技术被称为有利干扰的原因. Glossy[28]首先在网络洪泛中实现和验证了 CI 技术.

LWB[29](Low-power Wireless Bus)进一步将多跳低功耗无线网络转化成类似共享总线的架构，其中任意节点都是数据的接收者. LWB 将所有通信需求映射到快速 Glossy 洪泛上，并全局的调度每一个 Glossy 洪泛. 每个 Glossy 洪泛负责将数据包从一个节点发送到其他所有节点.

另一种提高消息分发速度的机制为流水线发包机制(PIP)[30]. Raman 等人提出的 PIP 采用流水线机制在同一条路径上使用不同信道传输大量数据. PIP 利用信道多样性来避免自干扰，每个转发节点都使用不同的接收信道. 流水线机制的关键在于，在任何时间，转发节点都在发送或接收数据包，使其能够达到最大的时间利用率.

进一步地，可以将 Glossy 与 PIP 结合起来，在消除信道冲突的同时，最大化时间利用率. 如图 3 所示，基于 Glossy 的 CI 技术和 PIP 的流水线机制，可以将包的传递构建成一棵传播树. 一个包被同一层的节点同时转发；同时，多个数据包能被不同层的节点使用不同信道并行传输. 在每个时间槽(即一个包的传输时间)，所有收到的包沿着树形结构传到下一层，即从父节点传到子节点.

Splash[31]为最近的结合 CI 技术与流水线机制的数据分发协议. Splash 将数据对象多次传输，并依赖于本地的恢复机制来获得高可靠性. 在最初的两轮分发中，Splash 将数据对象发送两次. 在第三

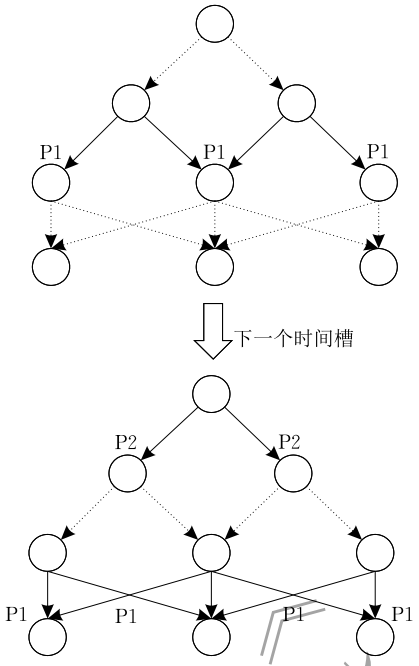

图 3 基于 CI 技术和流水线构建的流水线树[32]
(图中 P1 和 P2 分别代表第一个包和第二个包)

轮中,传输异或编码(XOR-encoded)的数据包. 每个编码后的数据包为 20 个随机选择的原始数据包的线性组合. 当一个节点收到一个编码后的数据包,并且已经收到用于编码该数据包的其它 19 个数据包,那么它可以恢复出其余一个丢失的原始数据包. 在快速流水线分发之后,本地恢复策略开始执行. 节点通过 CSMA/CA(Carrier Sense Multiple Access with Collision Avoidance)载波侦听多路访问协议向其邻居节点请求丢失的数据包.

Pando[32]在此基础上引入了喷泉码(Fountain Code),将需要分发的数据利用喷泉码机制编码为多个数据包,每个数据包携带不同的信息,一旦节点累积收到足够数目的数据包即可解码出原始数据. 在这种机制下,重传的数据包不需要是丢失的特定数据包,而可以是任意新编码的数据包,提高了数据分发的效率及可靠性.

总结

如何实现实时、可靠的带宽内控制,关键在于如何部署控制节点,以及如何将控制消息快速、可靠地从控制节点分发到普通传感节点上. 对于前者,现有的拓扑算法已经能够有效的解决,如连通支配集规划等. 对于后者,在传统传感器网络中一直是研究热点并有很多已有工作值得借鉴. 为了提高数据分发的可靠性,Glossy[28]利用了 CI 技术来分发数据,证明了 CI 技术能消除不必要的信道竞争,并提高了分发效率. 另一工作 PIP[30]采用流水线机制进行多信道通信,使得多跳路径中的相邻链接能同时通信,大大提高了数据分发速率. Splash[31]结合了 CI 技术与流水线机制,使得数据在快速并行的流水线上分发,极大地提高了信道利用率. 然而,由于同时发送数据的节点数目增多,CI 技术的可靠性受到很大影响,并且不同节点不同信道的通信质量都各不相同,使得数据分发的可靠性缺乏保障. Pando[32]引入了喷泉码机制提高了数据分发的可靠性,实现了无竞争的流水线. 与传统传感器网络不同,软件定义传感器网络有多个控制节点,各自负责一片区域内的控制消息分发,边界节点有可能收到来自多个不同控制节点的消息,如何消除干扰提高边界节点的可靠性有待研究. 此外,引入喷泉码机制在提高可靠性的同时,也增加了编解码的开销. 如何在实现快速可靠的同时,有效控制节点资源开销也值得进一步研究.

3.4 一致性

挑战

一致性问题有两类. 一类是在有多个控制节点的情况下,如何保证每个控制节点的逻辑视图的一致性;另一类是如何保证不同版本的网络配置的一致性. 为了保证网络的功能能够正常运行,维护网络中控制节点的逻辑视图的一致性,以及不同版本的网络配置之间的一致性是非常重要的.

关键技术

在因特网中,更新的错误通常由于中间件的错误配置所导致. 这个问题也存在于传感器网络中. 为解决该问题,一致性更新成为了软件定义网络中的热门研究话题.

最初,OpenFlow[2]的设计与实现基于网络中仅含有一个控制器的情况,然而,随着部署 OpenFlow 网络规模的增加,只依靠单个控制器已经不可行. 原因有三,其一,大幅增加的网络吞吐量使得单个中心控制节点无法承受;其二,更多的节点由于地理距离的原因将遭受更大的网络延迟;其三,单个控制器的计算能力有限,无法同时满足庞大网络节点的数据处理需求. 因此,OpenFlow 不得不引入含有多个控制器的软件定义网络.

多个控制器的软件定义网络中,每个控制节点都有全网的逻辑视图,为了使不同的控制节点能够协同工作,保持这些控制节点的一致性是十分重要的. HyperFlow[33]是一个基于 OpenFlow 的分布式事件驱动的控制层,它保证了控制层中控制节点逻辑视图的一致性. 当某个控制节点状态改变时,可以

发布一个状态改变的事件，其他的控制节点收到事件后，通过重现事件来更新状态，达到逻辑视图的一致. HyperFlow 在逻辑上是集中的，但是物理上是分布式的. 这让整个网络保持了一致性，并且有更好的可扩展性.

为了建立当前网络状态的逻辑视图，SDN-WISE[9]提出了一种拓扑管理层. 拓扑管理层可以访问所有协议层提供的应用程序接口(Application Interface，API)，从而控制所有协议层的行为，实现跨层的操作. 拓扑管理层首先收集每个节点的状态信息，例如，拓扑相关信息、剩余能量信息、链路的信噪比信息等，然后把信息发送给控制节点，形成整个网络的逻辑视图.

Reitblatt 等人[34]提出了一致性网络更新的概念，即在更新网络配置时，必须保证网络配置前后功能的一致性. 在他们的方案中，核心目标是保证数据包级别的一致性. 即每个经过该网络中的数据包都被一种全局的网络配置处理. 当网络更新发生时，在该网络中的所有数据包，或者使用此次网络更新之前的配置，或者使用此次网络更新之后的配置，不存在两者混合的情况. 而在原子更新机制中，所有交换机同时更新网络配置，这种方式容易产生多种网络配置混合的数据包，它们最终会被丢弃或者被发送至一个错误的目标.

为了确保数据包级别的一致性，Reitblatt 等人提出了一种两阶段更新机制. 在这种机制中，每个数据包中的特定标志位(如 VLAN 标志位)都记录了一个版本号. 为了将网络配置更新到下一个版本，它首先更改在网络边界上交换机的配置，以实现只允许通过记录有下一个版本号的数据包的交换机放在网络中. 接着，当该网络中不再含有上一个版本的数据包后，它再恢复交换机的配置，以启用新版本的网络配置.

总结

维护软件定义传感器网络中逻辑视图和网络配置的一致性，是使得网络功能得以正常运行，实现多个控制节点的协同工作的基础. 对于逻辑视图的一致性，现有的工作(如 HyperFlow[33]等)已能在链路质量稳定的因特网中较好的运行，然而这在链路质量不稳定的无线传感器网络中是不适用的. 对于网络配置的一致性，现有工作(如两阶段更新机制[34])虽然能够较好的实现网络配置的原子操作，排除中间状态的数据流，然而每次更新时需要消耗大量的时间排除网络内上一个版本的所有数据包，这在有实时更新网络配置需求的传感器网络中是不适用的. 因此，在链路质量不稳定的网络中，如何设计能够满足实时性需求的一致性网络配置更新系统，是一个值得进一步研究的问题.

3.5　安全性

挑战

在软件定义的传感器网络中，一个被破坏的程序将会导致整个网络的中断. 更进一步地，如果网络中的程序被攻击者控制，这将对整个网络造成无法预料的危害. 因此，如何抵御攻击者对传感器网络的攻击和控制是一个非常重要的问题.

关键技术

目前，在保证传感器网络中代码分发的安全性方面，已有一些初步的方案，如：传感器网络中节点认证分发程序的方案、在未来的家居传感器网络中搭建异常检测系统(Anomaly Detection System，ADS)的方案等.

利用软件定义的思想，传感器网络可以实现一种更加直观的网络流量监控程序. 控制节点通过编程进行细粒度的数据包检测，从而监测经过转发节点的网络流量. 这些由控制节点定期收集的网络流量数据，提供了一个实时集中的网络状态视图. 借助开放的应用程序接口，开发人员可以实现基于在线学习模型的异常流量检测算法，从而自动和快速的识别网络安全威胁[35].

基于 Merkle[36]树的数字签名是一种提供认证服务的方法. 在数字签名阶段，一个数据包将被分成若干子数据包，由不同节点向同一个目标节点发送，每个子数据包的散列值将作为 Merkle 树的叶子节点，非叶子节点的值由其两个子节点合并后的字符串的散列值得到，依次向根节点方向计算之后，最终得到的根节点的散列值被称为 Merkle Root[36]. 在验证阶段，我们首先从可信源处得到完整数据包的 Merkle Root，接着再从其他不可信源处接收子数据包，构造 Merkle 树. 将其他不可信源处得到的 Merkle Root 与可信源处得到的进行对比，就能够验证数据包是否被篡改. 实际上，接收节点并不需要等到所有的子数据包接收完毕后才开始验证完整性，对于每一个到达的子数据包，只需向可信数据源索要验证该子数据包所需的审计单元，即可验证该子数据包是否完整[37]. 此种验证方式虽然能够迅速的舍弃被破坏的子数据包并立即向发送方请求重传该分块数据包，但同时也大大增加了可信源与传感器节点之间的通信开销. 因此，在实际场景中将根据不同需求选择对应的验证方式.

最近,安全解决方案提供商 Radware 提出了一种在软件定义网络中实现抵御拒绝服务攻击的方案:DefenseFlowTM①. 之后,该公司还提供了简化且开源的版本:DefenseFlow、Defense4All 和 OpenDaylight. 在 DefenseFlow 方案中,控制节点搜集网络中流经转发节点的网络流量,通过与作为基准的网络流量模式进行对比,发现潜在的 DoS(Denial of Service)攻击. 在检测到潜在的 DoS 攻击威胁时,DefenseFlow 方案的网络流量分流机制会将可疑流量重定向到专用网络流量过滤中心(运行了 Radware DefensePro 的网络行为分析系统),执行细粒度的网络流量检查、签名分析和威胁消除等操作.

可信基随机阈值限速检测算法(Threshold Random Walk with Credit Based Rate Limiting, TRWCB)[38]是一种检测网络中主机是否受到蠕虫病毒感染的技术,它的思想是:蠕虫病毒为了在网络中迅速的扩散自己,会对全网进行连接请求测试,在一个网络中,如果某个主机连接请求的失败次数远大于其他主机,则认为这个主机受到了蠕虫病毒的感染. 具体地,对于每一个网络中的主机,TRWCB 算法都维护了一个初始连接列表(即 TCP SYN 标记),以表示它们还没有收到回复响应(即 SYN ACK 的响应). 当列表中维护的任何一个连接超过了响应时间的阈值,或者收到了 TCP 重置标记的响应时,TRWCB 算法将该连接从维护的初始连接列表中移除,并且增加当前主机的似然比(即增加了该主机被认为是受到蠕虫感染的可能性). 另一方面,当连接列表中的某一个连接收到了回复时,该主机对应的似然比就降低(即增加了是正常主机的可能性). 当某个主机的似然比超过了一定阈值时,这个主机就会被认为受到了蠕虫病毒感染.

恶意流量限速算法[39-40]是一种在主机端限制病毒继续扩散的机制,它所基于的观察是:在病毒传播的过程中,被感染的主机会在短时间内,试图连接到许多不同的服务器. 而未受感染的主机则更有可能重复尝试连接到最近访问的相同服务器. 每当有新的连接请求时,主机会检查其请求是否来自于称为"工作集"的列表,如果该请求属于该列表,主机正常执行该请求;反之,则将该请求添加到称为"延迟集"的列表中. 每隔时间 d 秒,主机都会从"延迟集"列表中选择一个连接重新进行请求. 然而,当发现"延迟集"列表中的元素数量超过一定阈值时,则发出受到病毒感染的警报. 实际上,限速检测法限制了一定时间内主机扫描全网主机的次数,如果超出阈值,则认为该主机很有可能受到了攻击.

最大熵检测算法[41]利用最大熵估计算法检测网络中的异常数据包. 首先,最大熵检测算法将网络中所有的数据包分为 2348 类,并且利用最大熵估计算法构造正常流量中每个数据包类的分布. 其中,数据包的分类从两个维度进行:数据包中包含的消息类别和数据包的目标端口号. 在该算法中,数据包中包含的消息分为 4 种:TCP、UDP、TCPSYN 和 TCPRST. 数据包中可包含的端口号有 587 个. 在检测阶段,最大熵检测算法将维护一个实时更新的滑动窗口,该窗口记录了数据包类别的分布情况. 将滑动窗口中记录的数据包类别分布与预先建立的正常流量的数据包类别分布进行比较,并计算它们的 KL 值(Kullback-Leibler). 当 KL 值超过一定阈值时,发出受到攻击的警报.

NETAD[42](Network Traffic Anomaly Detector)是一种基于规则过滤的异常网络流量检测算法. 许多攻击者在攻击之前会对目标主机发送某些特定的连接初始化数据包,因此,只需分析来源于某一个主机最开始的几个连续的数据包,便足以分析此次通信行为是否是恶意的. 对于数据包中开始的 48 个字节,NETAD 分别定义了 48 种不同的属性. 根据在一定时间内统计的不同属性出现的频率和时间间隔,NETAD 赋予每种属性相应的分值. 攻击者发送的初始化数据包会包含某些特殊的字节,这些字节在正常的网络流量中并不常见. 因此,数据包头中出现频率低的字节将赋予更高的值. 如果某个数据包的计算分值高于一定阈值,则认为该数据包是异常的.

Mehdi 等人[35]在基于 NOX 的控制器中实现了以上 4 种异常检测算法(即 TRWCB,恶意流量限速算法,最大熵检测算法和 NETAD),并在包含真实流量的数据集中测试了 4 种算法. 该数据集来自于部署在以下 3 个机器中所记录的网络流量:联网服务供应商(Internet Service Provider,ISP)的边界路由器、家庭网络路由器和在一间办公室的交换机. 实验结果表明,4 种算法都不能在互联网服务供应商边界路由中实现满意的异常检测结果,但是在家庭路由器和办公室的交换机中拥有高精确度的异常检测结果.

其中,对于 TCP 端口扫描攻击的检测,TRWCB 算法和基于主机的恶意流量限速算法的精度比另外两种算法都要高,这是因为 TRWCB 算法和恶意流

① DefenseFlow. http://www.radware.com/SiteCode/Templates/PressReleaseDetail.aspx?id=1631514

量限速算法都是针对主机的检测算法，并且它们都以异常连接请求的比例作为检测特征，这种方式特别适用于 TCP 端口扫描的攻击检测.

总结

在节点有限资源的情况下，如何有效抵御软件定义传感器网络中的攻击仍然是一个开放性的问题. 对于已有方案而言，基于 Merkle 树的数字签名方法虽然能够降低传感器网络中恶意数据分发的威胁，然而，随着所需的签名数量随着验证数据的增多呈指数级增长[37]，这在资源有限的传感器节点中是不切实际的. Mehdi 等人验证的 4 种异常检测系统虽然在软件定义网络中较为有效，但是在软件定义的传感器网络中未必适用. 比如，链路质量不稳定的无线传输使得仅以链接是否保持来判断异常攻击是不适用的，链路质量的估计也应当作为检测特征考虑在内.

4　分　类

软件定义传感器网络相对于传统的传感器网络具有一大优势，即能利用规则的分发与节点流表的配置实现不同的软件定义功能. 因此，本文从软件定义功能的角度对现有的技术方案进行了分类. 如表 3 中所示，本文将现有的解决方案根据其功能分类为软件定义感知、软件定义路由、软件定义测量、软件定义调试和其他，并总结了其分别使用的技术. 值得注意的是，本节所述分类立足于软件定义功能的角度，它与第 3 节所述应对挑战所需的关键技术角度有所不同. 因此本节分类下各个方案所使用的技术与第 3 节所述的技术并不完全对应. 本文旨在从更多的维度总结当前软件定义传感器网络在架构设计方面的工作，从而挖掘出更多可能的研究空间.

表 3　现有的解决方案

已有方案	功能	使用的技术
QDiff[43]	软件定义感知	动态链接与加载技术
Tenet[12]	软件定义感知	虚拟机技术
Elon[44]	软件定义感知	动态链接与加载技术、应用组件与系统组件[45-47]分离技术
Sensor OpenFlow[7]	软件定义路由	控制器与节点之间的接口设计
TinySDN[8]	软件定义路由	控制器与节点之间的接口设计
SDN-WISE[9]	软件定义路由	状态机设计技术
Spectrum Monitoring[48]	软件定义测量	频谱监控、硬件设计
OpenSketch[49]	软件定义测量	三段式哈希管道、资源分配
TPP[50]	软件定义测量	测量任务接口设计、支持测量指令硬件设计
TinySDM[51]	软件定义测量	动态链接与加载技术、类 C 语言设计
Declarative Tracepoints[13]	软件定义调试	动态注入语言设计
Dylog[14]	软件定义调试	动态注入带宽内日志收集与同步、调试功能[45-46]抽象
Quanto[56]	软件定义调试	系统调用拦截技术
Deluge[57]	其他	启动引导器设计、可靠的分发技术
Stream[58]	其他	可靠的分发技术
R2[59]	其他	增量编程技术
SD-IoT[3]	其他	基于城市感知的物联网架构

4.1　软件定义感知

感知是传感器网络中最重要的任务之一. 借助软件定义感知，传感器网络可以通过动态地配置节点来实现不同的感知应用. 例如，通过动态地修改传感器网络的配置，节点能够按需调整感知周围环境参数的周期，并在探测到特定事件发生时发送通知信息. 可见，为了实现高效的软件定义感知，节点应当具备增量、动态的更新程序的能力. 现有工作已经在实现软件定义感知功能方面有所尝试：虚拟化技术（如 3.1 节所述）[12]、动态链接与加载以及动态代码注入技术[47-48].

QDiff[47] 是一种在动态配置传感器节点程序时保证功能完整性的方法. 其特点在于，在保持程序版本更新前后最大相似性的同时，利用克隆检测的方法计算最小的补丁程序（patch）大小. 大多数现有的重编程机制未能处理由于变量插入或变量删除操作导致的全局变量偏移问题，因此，QDiff 查找程序版本更新前后相似的函数和变量，从而重新组织函数和变量的布局，以保持更新前后两个程序之间最大的相似性，这个方法大大的减少了补丁程序的大小.

Tenet[12] 网络架构包括底层的普通节点和上层的控制节点. Tenet 网络结构限制了上层的多节点融合，允许底层的普通节点处理本地的感知数据，从而精简了应用的开发，提高了普通节点上的软件复用率. Tenet 使用了一种新的 TaskLet 库来描述这些应用.

例如，想要实现一个任意节点采集的温度超过 50°F 就报警的功能时，可以按照表 4 所示格式编写：

表 4　样例程序[12]

```
Sample(1000 ms,1,REPEAT,ADC10,TEMP)→LEQ(A,TEMP,50)
→DeleteActiveTaskIf(A)→Send()
```

但该程序有一个严重的缺陷，即必须在节点上安装支持 Tenet 脚本语言的虚拟机才能运行，且在运行时是无法动态加载软件库的. 因此，在编译程序时必须将所有可能用到的 TaskLet 库静态载入.

为了解决这个问题,我们提出了可以在传感器网络中模块化重编程的系统:Elon[48]. Elon 系统中的一个核心概念是可替换组件,该组件定义了传感器的感知函数. 借助 Elon 系统,传感器节点通过修改可替换组件就可以动态的改变传感器节点上的代码.

如图 4 所示为 Elon 对 TinyOS 的拓展. 其中,可替换组件由可替换数据和可替换代码构成,由用户在应用程序中进行声明,表示该组件在将来可能进行更换或更新. 同时,在应用程序中也需要连接可能使用的 TinyOS 内核组件(例如,CTP[49],Drip[50],FTSP[51]). Elon 设计的核心思想是,通过区分应用的可替换部分与 TinyOS 内核部分,使得在将来更新应用时,只需要重新编写小而简单的可替换组件,而无需分发大而复杂的 TinyOS 内核.

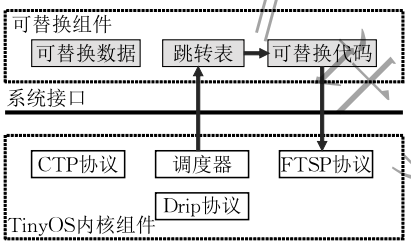

图 4 Elon 对 TinyOS 的拓展[48]

为了说明 Elon 的工作过程,我们称第一次烧写到节点上的程序为基础版本,后续更新组件后的版本为更新版本. TinyOS 内核组件包含在基础版本中以供将来使用. 对于基础版本,内核组件和可替换组件都由硬件烧录器烧写到程序闪存. 随后,节点进行硬件重启后,才能正常工作. 对于更新版本,只有可替换组件被生成并分发到每个节点上. 当节点接收到新代码时,只需要重新启动该替换组件来执行新代码.

Elon 允许应用程序定义“系统”接口(通过“@system()”注释)来访问 TinyOS 内核服务(类似于类 Unix 系统中的系统调用). 这些接口定义了可替换组件和 TinyOS 内核之间的边界. 对于在系统接口中定义的 downcall(即 TinyOS 命令),Elon 将更新版本中的引用(即,可替换组件中的命令调用)重新定位到基础版本中的相应地址(即,内核组件中的命令实现). 对于在系统接口中定义的 upcall(即 TinyOS 事件),Elon 维护系统跳转表以将基础版本中的引用(即,内核组件中的事件调用)重定向到相应的地址(即,可替换组件).

通过区分 TinyOS 内核组件和可替换组件,Elon 能够显著减少传输的代码大小. 可替换组件更新后,Elon 只需要重新启动该替换组件. 这种部分重启机制能够避免 TinyOS 内核数据的丢失. 此外,通过将可替换部件置于 RAM 中(对于冯诺依曼架构),Elon 避免了闪存写入,从而显著延长了 TelosB 节点可编程寿命.

4.2 软件定义路由

路由也是传感器网络中的一项重要任务,特别是对于多跳的 Mesh 网络而言,具备根据网络的拓扑以及链路质量的变化动态配置路由策略的能力是非常重要的. 当前已经有一些方案尝试实现了软件定义路由的功能:与传统软件定义网络类似,包括将控制平面和数据平面分离的方案[7-8],以及利用有限状态机定义传感器节点行为的方案[9]等.

Luo 等人[7]提出了一种应用在传感器网络的软件定义架构:Sensor OpenFlow. 此架构与软件定义网络架构类似,分离了控制层与数据层,能够对包的路由策略实现更灵活的配置. 具体地,每个传感器节点上都设置了流表,包含两列属性:匹配规则以及匹配行为. 匹配规则用于匹配在本地生成或者转发的数据包;匹配行为定义了对满足匹配规则的数据包采取的行为. 例如,发向指定的端口或者丢弃. 控制层的控制器通过自定义的 OpenFlow 协议与传感器节点交互. Sensor OpenFlow 通过借鉴 OpenFlow 的设计思想,解决了在传感器网络场景中的一些特定的问题:如何创建流、如何减少控制流、如何进行数据聚合等网内数据处理.

TinySDN[8]是基于 TinyOS 平台实现的软件定义网络. 如图 5 所示,在 TinySDN 架构中,有两种传感器节点:软件定义的传感器节点和软件定义的控制节点. 软件定义的传感器节点为配置了流表的普通传感器节点. 软件定义的控制节点为执行控制任务的节点,通常实现为与服务端相连的传感器节点. 该服务端上实现了控制层的逻辑,并通过与之相连的传感器节点与网络中其它节点交互. 与 Sensor OpenFlow 不同的是,TinySDN 在网络中布置了多个控制节点,以减少传输控制流的开销. 在这样的架构里,每个软件定义的传感器节点都可以使用类似 CTP(Collection Tree Protocol)的多跳传输协议,来发现就近的控制节点.

为了进一步降低传输控制流的开销,SDN-WISE[9]将一部分控制逻辑放在本地节点上,这些控制逻辑仅需要依据本地信息决定对包的处理,而无需与控制节点进行交互. 在 TinySDN 中,传感器节

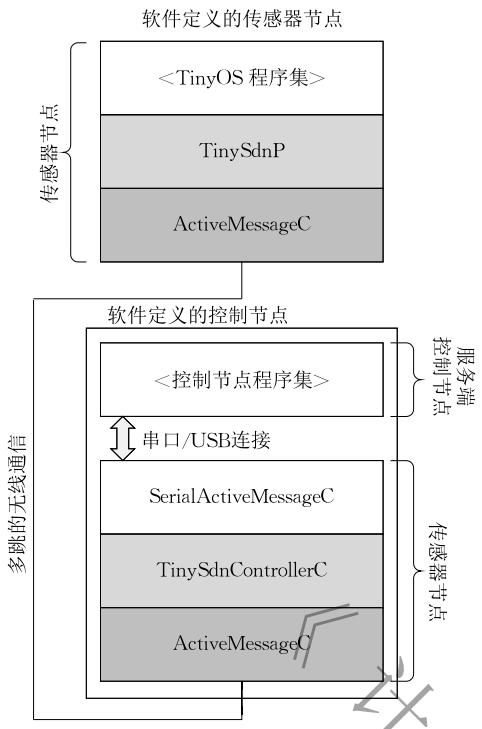

图 5　TinySDN 节点架构[8]

点是无状态的，而在 SDN-WISE 中，传感器节点是有状态的. SDN-WISE 将对包的处理逻辑编码成有限自动状态机，因而在处理包时无需控制节点参与.

在 SDN-WISE 中，节点所有行为都被定义在 3 种数据结构中：WISE 状态数组、接收序号数组和 WISE 流表. 与大部分软件定义网络类似，这些数据结构中的信息都来自运行在服务器上控制节点. 借助这种方式，控制节点制定网络策略，普通传感器节点实现并执行该策略.

WISE 状态数组为包含节点状态信息的数据结构. 在任意时刻，传感器节点根据控制策略及本地信息决定自己的当前状态. 接收序号数组规定了节点需要做出处理的节点序号. 由于 WSN 的广播属性，节点会收到来自很多其它节点包的干扰，接收序号数组能够使节点对这些包有选择的进行处理. 当节点判定收到的包需要被处理时，将根据自己的 WISE 流表来决定要执行的操作. 流表的表项由匹配规则和行为构成. 匹配规则包含对包内任意内容的匹配，或对节点状态的匹配. 满足匹配规则的包，节点将根据流表中对应项的行为，执行相应的操作.

SDN-WISE 能够有效降低传感器节点与控制节点之间的交互，从而减少传输开销. 然而，引入状态将对包的处理逻辑实现在本地，将使得流表体积成倍增长，并使得匹配过程更加费时，从而增加了包的传输时延. 如何减少流表表项及提高匹配速度，值得进一步研究.

4.3　软件定义测量

在传统的传感器网络中，有各种各样的网络测量任务，如测量网络延迟、丢包率等. 然而，为不同的任务分别编写不同的代码会消耗许多额外的人力、物力. 因此，如何从纷繁复杂的测量任务中抽象出公共的模块，并利用软件定义的思想方便、快捷的部署多种多样的测量任务，正是软件定义测量所要达成的目标. 现有工作有的基于高定制化的硬件[52]，有的基于软件定义网络[53-54]，也有的基于传感器网络抽象的软件定义测量架构[55].

Pfammatter 等人[52]提出了一种用于监测大规模宽带频谱的软件定义的传感器架构. 该架构基于非常廉价的商用成品传感器硬件，整体价格低于 100 美元. 因此，它能够利用群智感知的方法，在更大规模的区域和更长的时间跨度上监控频谱的使用状况.

在该架构中，主要的组件有以下 3 部分：

(1) 传感器. 由 4 个主要的组件构成：单板计算机(Single-Board Computer, SBC)平台、RTL-SDRUSB 转换器、天线和温度传感器. RTL-SDRUSB 转换器与连接它的天线作为射频接收装置. SBC 平台作为计算主机，主要负责射频接收和温度传感；

(2) 控制器. 控制器是一个访问分布式传感器的命令与控制中心，它主要负责给每个节点分配合适的监控任务，并跟踪节点的状态；

(3) 收集器. 收集器是系统的数据收集单元，它们一般负责大规模的数据预处理.

作者在 SBC 上实现了常见频谱的分析，传感器管理员可以远程访问并配置所有参数. 例如，管理员可以通过远程配置频谱监视器参数的方式调整频率纠正器的计算.

除了监控频谱的使用状况，在传感器网络测量中还有许多其他的度量. 例如，能量利用率、链路丢包率等. 在传统的传感器网络测量技术中，测量任务是定制的，即针对不同的测量任务，需要分别编写特定的程序. 因此，在已部署的网络中，更换测量任务非常困难. 为了能够在传感器网络中灵活的配置测量任务，我们可以从因特网的软件定义测量中得到启示.

OpenSketch[53]是一种因特网中的软件定义测量架构，对传感器网络中的软件定义测量发展起到了十分重要的指导作用. OpenSketch 将控制平面和数据测量平面分开. 在数据测量平面上，OpenSketch

将测量任务抽象为 3 个步骤，并提供了一个简单的三段式管道(哈希、筛选和计数)，从而在商业交换机实现并支持多种测量任务. 在控制平面上，OpenSketch 可以自动为不同的测量任务配置管道，并为其分配资源.

TPP(Tiny Packet Program)[54] 同样为传感器网络中实现软件定义测量带来了重要启示. TPP 提供一种简单可配置的接口，使得汇聚节点能够将测量指令附在数据包里，从而在数据平面上直接查询交换机的内存数据. 特别地，指令附加在数据包的头部，包括读写数据包内容或交换机内存等简单操作. 在 TPP 中，指令与执行结果都附加在包头中. 同样地，TPP 提供的简单指令，无法满足传感器网络中复杂多变的测量任务需求.

软件定义网络测量的兴起为传感器网络测量技术提供了大量理论与实践基础. 我们在此基础上，结合传感器网络测量场景多变，任务复杂多样的特点，提出了首个基于软件定义思想的传感器网络测量架构 TinySDM[55]，使测量任务与应用相互独立，在不干扰应用正常运行的情况下灵活配置测量任务.

TinySDM 的总体架构如图 6 所示，TinySDM 的工作过程分为两个阶段：初始化阶段及测量阶段. 在初始化阶段中，根据配置文件(Hook.conf)将一系列钩子(Hook)插入到原始程序的相应位置上，这些位置即为未来可能执行测量任务的位置. 当程序运行到钩子所在位置时，能跳转执行相应的策略任务. 随后，插入钩子的程序映像将被烧写到网络中的节点上. 在测量阶段中，程序员应用类 C 语言 TCL(TinyCode Language)来编写和定制自己的测量任务，并使用 TinyCode 编译器生成二进制任务文件. 然后，TinySDM 将该二进制任务文件嵌入到控制包中，并发送给所有节点. 由于只需要传送测量任务的二进制代码，TinySDM 极大的减少了分发任务的传输开销.

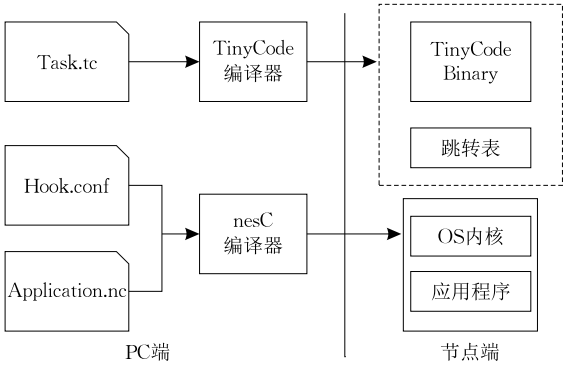

图 6 TinySDM 总体架构[55]

4.4 软件定义调试

在传感器网络中，软件的调试与诊断非常困难，它们通常需要多轮检测以精确定位出现问题的原因. 由于传感器网络节点资源有限，对所有变量监控是不可行的，传统传感器网络的调试与诊断缺少系统状态的可见性，难以在正确的时刻监控必要的变量. 软件定义调试的功能允许用户动态地向节点应用程序插入监测点以监控必要的变量，从而按需配置调试方案，有效提高了调试与诊断的效率[13-14].

DT(DeclarativeTracepoints)[13] 允许用户对运行的程序动态的插入一组行为相关的检测点或追踪点，对已部署的网络而言，这一点非常重要. 设置追踪点不需要修改源代码，它们仅需通过声明性的、类 SQL 语言的 TraceSQL 进行编程即可. TraceSQL 提供三种常见的调试操作，包括读取或设置内存变量值、调用函数以及终止或跳出线程.

表 5 给出用 TraceSQL 实现的实例 StackGuard[43]. 该实例用于检测由于缓冲区溢出造成的栈损坏(例如某函数的返回地址被重写). StackGuard 利用安全字(Canary Word)来检测所有的缓冲区溢出漏洞. 具体地，StackGuard 修改函数序言(Function Prologue)和尾声(Function Epilogue)来插入安全字，在函数返回时检测安全字是否被修改. 如果安全字被修改，则认为函数返回地址被重写. 这是因为，缓冲区溢出在空间上是连续性的，利用栈空间上的缓冲区溢出在修改函数返回地址之前，会先修改安全字. 如表 5 所示的示例代码中，对需要检测的函数(即 app.c 中的 crashNode()函数)做了两次注入. 第一次注入位于函数序言，另一次位于函数尾声，关键字 BEFOREPROLOGUE 及 AFTEREPILOGUE 指定了插入点的位置. 在第一个注入点处插入安全字@canarynumber，在第二个注入点处检

表 5 TraceSQL 实现 StackGuard[43]

```
1.  INTEGER @canarynumber=getRand();
2.  INTEGER @testnumber;
3.  TRACE crashNode() BEFOREPROLOGUE
4.  FROM app.c EXECUTE
5.  {
6.      push integer(@canarynumber);
7.  }
8.  TRACE crashNode() AFTEREPILOGUE
9.  FROM app.c EXECUTE
10. {
11.     @testnumber=pop integer();
12.     IF (@testnumber !=@canarynumber) {
13.         BREAK;
14.  }
15. }
```

查该安全字是否被修改.可以看到,TraceSQL 易于编程且实现高效,在本例中实现 StackGuard 仅用了 15 行代码.

开发者借助 DT 中的 TraceSQL 就能够用很少的代码实现以往功能孤立的调试技术,如 StackGuard[43]、EnviroLog[44]、NodeMD[45] 和 Sympathy[46]的核心功能.其中,EnviroLog 为事件驱动的应用提供事件记录及重放服务;NodeMD 用于诊断传感器网络节点级别的应用异常问题,主要关注于三种故障:栈溢出、死锁和活锁;Sympathy 通过在节点收集多种信息用于分析诊断,包括网络状态信息、网络流量信息以及节点自身的测量信息.这些调试技术均可以通过 TraceSQL 来配置实现.

Dylog[14]用于动态地向已部署的系统中注入日志记录语句,且可以根据需求动态的更改日志记录语句.与 DT 不同的是,Dylog 解决了高效的日志存储和准确的节点日志同步的问题.首先,Dylog 能够动态地插入或删除日志记录的语句,从而实现灵活的交互式调试功能.其次,Dylog 集成了一个高效的存储系统和日志收集方案,用于记录和传输日志消息.最后,Dylog 采用了轻量级数据驱动的方法进行时间同步和时间重建.如图 7 所示,在 PC 端,DyDiff 比较两个版本的代码之间的差异,产生基于代码差异的补丁程序(Patch),之后,该补丁程序通过代码分发协议,发送到每一个节点上.在节点端,DyAgent 负责安装接收到的补丁程序并执行新的代码.日志消息则通过高效的存储方案存储在 Flash 上.一旦有请求产生,记录的消息通过日志收集协议,被收集回 PC.在 PC 端,记录的消息以正确的格式和同步的时间戳重建.

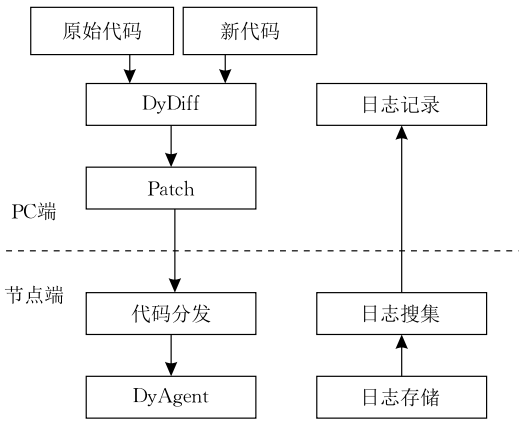

图 7　Dylog 总体架构[14]

Quanto[56]通过对设备驱动的修改,拦截了功能模块(如 CPU,Flash 等)能耗状态改变的所有事件,同时记录整个节点能耗改变 ΔE 和消耗时间 Δt.如此,每次有状态改变时可得到一个等式.等式左边是 ΔE,等式右边是每个模块各自当前状态能耗系数乘以 Δt 的累加.当等式足够多,达到线性方程组可解约束时,即可解出每个模块对应状态的能耗.由此,Quanto 通过在 TinyOS 中修改系统调用来监控能耗状态,并结合状态能耗系数,即可追踪各模块的细粒度能量消耗.

4.5　其　他

本文已经提到了许多不同的软件定义功能,如软件定义感知、软件定义路由、软件定义测量和软件定义调试.借助软件定义的技术,用户不仅能够轻松地定制新策略,还能动态更新策略,快速验证不同策略在已部署网络中的效果.还有一些软件定义传感器网络的其他工作,它们有的是实现软件定义传感器网络功能的支撑技术,有的是搭建了包含部分软件定义传感器网络功能的框架[3].

Deluge[57]是基于 TinyOS 的重编程方法.在 Deluge 中,为了修改某个应用程序,需要将修改后的应用程序和整个操作系统都重新编译,并覆盖传感器节点上的操作系统.为了避免在重编程过程中可能引发的误擦除问题,Deluge 提供了一个小型引导加载程序 TOSBoot,它考虑了与此相关的一系列问题,以便于开发者编程.

TOSBoot 作为一个独立的程序,任何由 TinyOS 程序遗留下来的非易失性状态,都不会影响它的执行.在 TOSBoot 中,它只能通过物理接入的方式安装,且没有使用任何中断向量(除了重置电源这一物理中断),并始终以禁止中断的状态运行.TOSBoot 使用了非易失性存储器传递参数,这些参数包括:在启动过程中连续中断的次数、是否需要对微处理器编程、对微处理器编程的二进制文件在外部存储器的偏移.借助非易失性存储器,确保了节点即使发生意外中断或者启动过程中的电源故障等问题,也能正确操作.

在启动过程中,TOSBoot 会检测是否需要对微处理器编程.如果需要,它会擦除程序存储器,并向其中写入新的二进制文件.当写入完成后,TOSBoot 重置编程位,跳转到程序的第一条指令.如果不需要,则直接跳转到需要执行程序的第一条指令.

现在已经有许多技术都能够提升 Deluge 的性能.Deluge 把重编程协议和应用程序绑定,作为整个程序镜像进行更新.而 Stream[58]将重编程协议与具有监听程序更新能力的应用程序分割为两个

程序镜像,这使得以后每次更新时只需要更新应用程序,从而减少了数据传输量,降低了延迟.具体地,Stream 先给所有的节点预先安装重编程协议映像,即 Stream-RS 组件. Stream-RS 组件是基于 Deluge 的,使用三次握手完成多跳代码分发.另一部分是具有更新能力的应用映像,即 Stream-AS. Stream-AS 是通用的,在任何 TinyOS 应用程序中使用时,只需要插入两行代码.在一个网格拓扑的真实测试平台上,Stream 与 Deluge 做了对比实验,主要测量了传输的字节数(与能耗密切相关)和延迟.实验表明 Stream 相比 Deluge,对网格拓扑重编程减少了 63%到 98%的传输时间,传输的字节数减少了 75%到 132%.

然而,对于实际中复杂的应用来说,Stream 依然不够有效.为此,我们提出了增量重编程技术 R2[59],它能够更进一步地大幅度减少它在网络中的代码传输量.与以前传输整个新的程序代码不同的是,R2 比较新旧代码之间的差异,然后将差异部分的代码片分发到所有的节点上,节点根据收到的差异片段,就能够结合已有的代码重构新的程序.但重编程技术的高灵活性往往伴随着高开销(包括,分发差异代码片的传播开销、存储器的 I/O 开销以及系统重启的开销等)[60].

Liu 等人[3]提出了一种软件定义物联网的架构(Software-Defined Internet of Thing,SD-IoT),该架构用于城市的感知.虽然目前物联网设备的部署为城市的感知提供了基础,但由于每个应用都需要部署和管理各自的感知平台,让维护费变得高昂.同时,感知平台、网关、服务器等之间的高度耦合,增加了需求改变时的修改成本.此外,管理员无法动态的控制数据采集及传输,让资源无法有效的利用.SD-IoT 把感知应用与物理设备剥离开,中心控制节点对设备进行统一管理,为城市感知的各种应用提供数据采集、数据传输、和数据处理的接口.

5 总结和未来工作

本文调研了现有的软件定义传感器网络架构的设计,列举了面向通用、高效的软件定义传感器网络的主要挑战,分别是通用的软件定义架构、资源的有效分配、快速更新与实时的带宽内控制、一致性、安全性.为了解决这些挑战,本文总结了许多有用的技术并分析了这些技术的不足.此外,根据抽象出来的功能,本文还从软件定义感知、软件定义路由、软件定义测量、软件定义调试、其他五个方面对软件定义传感器网络进行了的分类,并总结了其分别使用的技术.在未来,软件定义传感器网络架构将为现有网络中的软件开发带来极大的进步与快速的技术革新.

目前,软件定义传感器网络的研究和发展仍然是当前研究的热点,有大量的开放性问题有待更加深入的研究.以下列举出几点未来可能的发展方向:

(1)便捷的编程模型.软件定义传感器网络架构的一大特点就是软件更新的灵活与实用,如何设计一个便捷的编程模型是当前研究持续的焦点;

(2)权衡能耗、灵活性和编程便捷性的设计.能耗低、灵活性强和编程便捷是不可兼得的三点,如何在不同的应用场景中权衡以上三点是一个值得研究的问题;

(3)无线重编程与远程调试的结合.无线重编程与远程调试是编程中两个十分有用的技术,目前已有大量的工作致力于无线重编程和远程调试,如何将这些工作合并到一个未定义的软件定义传感器网络架构中,也是一个需要思考的问题.

可以预见,有了通用、高效的软件定义传感器网络架构的强力支持,未来的传感器网络的软件开发将会大大的简化.

参 考 文 献

[1] Estrin D, Govindan R, Heidemann J, Kumar S. Next century challenges: Scalable coordination in sensor networks// Proceedings of the 5th Annual International Conference on Mobile Computing and Networking (MobiCom). Seattle, USA, 1999: 263-270

[2] McKeown N, Anderson T, Balakrishnan H, et al. OpenFlow: Enabling innovation in campus networks. ACM SIGCOMM Computer Communication Review, 2008, 38(2): 69-74

[3] Liu J, Li Y, Chen M, et al. Software-defined Internet of Things for smart urban sensing. IEEE Communications Magazine, 2015, 53(9): 55-63

[4] Hu F, Hao Q, Bao K. A survey on software-defined network and OpenFlow: From concept to implementation. IEEE Communications Surveys & Tutorials, 2014, 16(4): 2181-2206

[5] Zeng D, Miyazaki T, Guo S, et al. Evolution of software-defined sensor networks//Proceedings of the IEEE 9th International Conference on Mobile Ad-Hoc and Sensor Networks (MSN). Dalian, China, 2013: 410-413

[6] Jagadeesan N A, Krishnamachari B. Software-defined networking paradigms in wireless networks: A survey. ACM Computing Surveys, 2015, 47(2): 27:1-27:11

[7] Luo T, Tan H P, Quek T Q. Sensor OpenFlow: Enabling software-defined wireless sensor networks. IEEE Communications Letters, 2012, 16(11): 1896-1899

[8] Oliveira B T, Gabriel L B, Margi C B. TinySDN: Enabling multiple controllers for software-defined wireless sensor networks. IEEE Latin America Transactions, 2015, 13(11): 3690-3696

[9] Galluccio L, Milardo S, Morabito G, Palazzo S. SDN-WISE: Design, prototyping and experimentation of a stateful SDN solution for WIreless SEnsor networks//Proceedings of the 34th IEEE Conference on Computer Communications (INFOCOM). Hong Kong, China, 2015: 513-521

[10] Gante A, Aslan M, Matrawy A. Smart wireless sensor network management based on software-defined networking//Proceedings of the 27th Biennial Symposium on Communications (QBSC). Kingston, Canada, 2014: 71-75

[11] Liu Y, He Y, Li M, et al. Does wireless sensor network scale? A measurement study on GreenOrbs//Proceedings of the 30th IEEE Conference on Computer Communications (INFOCOM). Shanghai, China, 2011: 873-881

[12] Gnawali O, Jang K Y, Paek J, et al. The Tenet architecture for tiered sensor networks. ACM Transactions on Sensor Networks, 2010, 6(4): 34:1-34:44

[13] Cao Q, Abdelzaher T, Stankovic J, et al. Declarative tracepoints: A programmable and application independent debugging system for wireless sensor networks//Proceedings of the 6th ACM Conference on Embedded Network Sensor Systems (SenSys). New York, USA, 2008: 85-98

[14] Dong W, Huang C, Wang J, et al. Dynamic logging with dylog for networked embedded systems//Proceedings of the 11th Annual IEEE International Conference on Sensing, Communication, and Networking (SECON). Singapore, 2014: 381-389

[15] Liu H, Roeder T, Walsh K, et al. Design and implementation of a single system image operating system for ad hoc networks //Proceedings of the 3rd International Conference on Mobile Systems, Applications, and Services (MobiSys). Seattle, USA, 2005: 149-162

[16] Levis P, Culler D. Maté: A tiny virtual machine for sensor networks. ACM Sigplan Notices, 2002, 37(10): 85-95

[17] Levis P, Madden S, Polastre J, et al. TinyOS: An operating system for sensor network//Weber W, Rabaey J M, Aarts E eds. Ambient Intelligence. Berlin Heidelberg, Germany: Springer, 2005: 115-148

[18] Han C C, Kumar R, Shea R, et al. A dynamic operating system for sensor nodes//Proceedings of the 3rd International Conference on Mobile Systems, Applications, and Services. Seattle, USA, 2005: 163-176

[19] Marrón P J, Gauger M, Lachenmann A, et al. FlexCup: A flexible and efficient code update mechanism for sensor networks//Proceedings of the 3rd European Conference on Wireless Sensor Networks. Zurich, Switzerland, 2006: 212-227

[20] Dunkels A, Finne N, Eriksson J, Voigt T. Run-time dynamic linking for reprogramming wireless sensor networks//Proceedings of the 4th International Conference on Embedded Networked Sensor Systems (Sensys). New York, USA, 2006: 15-28

[21] Ekman M, Thane H. Dynamic patching of embedded software //Proceedings of the 13th IEEE Real Time and Embedded Technology and Applications Symposium (RTAS). Bellevue, USA, 2007: 337-346

[22] Zeng D, Li P, Guo S, et al. Energy minimization in multi-task software-defined sensor networks. IEEE Transactions on Computers (ToC), 2015, 64(11): 3128-3139

[23] Petrioli C, Nati M, Casari P, et al. ALBA-R: Load-balancing geographic routing around connectivity holes in wireless sensor networks. IEEE Transactions on Parallel and Distributed Systems, 2014, 25(3): 529-539

[24] Li S, Zhao S, Wang X, et al. Adaptive and secure load-balancing routing protocol for service-oriented wireless sensor networks. IEEE Systems Journal, 2014, 8(3): 858-867

[25] Huang R, Chu X, Zhang J, et al. Energy-efficient monitoring in software defined wireless sensor networks using reinforcement learning: A prototype. International Journal of Distributed Sensor Networks, 2015, 11(10): 1-12

[26] Jayashree P, Princy F I. Leveraging SDN to conserve energy in WSN-An analysis//Proceedings of the 17th International Conference on Signal Processing, Communications and Networking. Chicago, USA, 2015: 1-6

[27] Cheng X, Huang X, Li D, Du D Z. On the construction of connected dominating set in ad hoc wireless networks//Proceedings of the 2006 International Wireless Communications and Mobile Computing (IWCMC). New York, USA, 2006: 183-190

[28] Ferrari F, Zimmerling M, Thiele L, Saukh O. Efficient network flooding and time synchronization with Glossy//Proceedings of the 10th International Conference on Information Processing in Sensor Networks (IPSN). Chicago, USA, 2011: 73-84

[29] Ferrari F, Zimmerling M, Mottola L, Thiele L. Low-power wireless bus//Proceedings of the 10th ACM Conference on Embedded Network Sensor Systems (Sensys). Toronto, Canada, 2012: 1-14

[30] Raman B, Chebrolu K, Bijwe S, Gabale V. PIP: A connection-oriented, multi-hop, multi-channel tdma-based mac for high throughput bulk transfer//Proceedings of the 8th ACM Conference on Embedded Networked Sensor Systems (Sensys). Zurich, Switzerland, 2010: 15-28

[31] Doddavenkatappa M, Chan M C, Leong B. Splash: Fast data dissemination with constructive interference in wireless sensor networks//Proceedings of the 10th USENIX Symposium on Networked Systems Design and Implementation (NSDI). Seattle, USA, 2013: 269-282

[32] Du W, Liando J C, Zhang H, et al. When pipelines meet fountain: Fast data dissemination in wireless sensor networks //Proceedings of the 13th ACM Conference on Embedded

Networked Sensor Systems (Sensys). Seoul, South Korea, 2015: 365-378

[33] Tootoonchian A, Ganjali Y. HyperFlow: A distributed control plane for OpenFlow//Proceedings of the 2010 Internet Network Management Conference on Research on Enterprise Networking (INM/WREN). Berkeley, USA, 2010: 1-6

[34] Reitblatt M, Foster N, Rexford J, et al. Abstractions for network update//Proceedings of the ACM SIGCOMM 2012 Conference on Applications, Technologies, Architectures, and Protocols for Computer Communication. New York, USA, 2012: 323-334

[35] Mehdi S A, Khalid J, Khayam S A. Revisiting traffic anomaly detection using software defined networking//Proceedings of the 14th International Symposium on Recent Advances in Intrusion Detection. California, USA, 2011: 161-180

[36] Hyun S, Ning P, Liu A, et al. Seluge: Secure and DoS-resistant code dissemination in wireless sensor networks//Proceedings of the 7th International Conference on Information Processing in Sensor Networks (IPSN). Washington, USA, 2008: 445-456

[37] Buchmann J, Dahmen E, Klintsevich E, et al. Merkle signatures with virtually unlimited signature capacity//Proceedings of the 5th International Conference on Applied Cryptography and Network Security (ACNS). Zhuhai, China, 2007: 31-45

[38] Schechter S E, Jung J, Berger A W. Fast detection of scanning worm infections//Proceedings of the Seventh International Workshop on Recent Advances in Intrusion Detection. Heidelberg, Berlin, Germany, 2004: 59-81

[39] Twycross J, Williamson M M. Implementing and testing a virus throttle//Proceedings of the 12th USENIX Security Symposium. Washington, USA, 2003: 285-294

[40] Williamson M M. Throttling viruses: Restricting propagation to defeat malicious mobile code//Proceedings of the 18th Annual Computer Security Applications Conference. Las Vegas, USA, 2002: 61-68

[41] Gu Y, McCallum A, Towsley D. Detecting anomalies in network traffic using maximum entropy estimation//Proceedings of the 5th ACM SIGCOMM Conference on Internet Measurement. Berkeley, USA, 2005: 32-32

[42] Mahoney M V. Network traffic anomaly detection based on packet bytes//Proceedings of the 2003 ACM Symposium on Applied Computing. Florida, USA, 2003: 346-350

[43] Cowan C, Pu C, Maier D, et al. StackGuard: Automatic adaptive detection and prevention of buffer-overflow attacks//Proceedings of the 7th USENIX Security Symposium. San Antonio, USA, 1998: 63-78

[44] Luo L, He T, Zhou G, et al. Achieving repeatability of asynchronous events in wireless sensor networks with envirolog//Proceedings of the 30th IEEE Conference on Computer Communications (INFOCOM). Barcelona, Spain, 2006: 1-14

[45] Krunic V, Trumpler E, Han R. NodeMD: Diagnosing node-level faults in remote wireless sensor systems//Proceedings of the 5th International Conference on Mobile Systems, Applications and Services. New York, USA, 2007: 43-56

[46] Ramanathan N, Chang K, Kapur R, et al. Sympathy for the sensor network debugger//Proceedings of the 3rd International Conference on Embedded Networked Sensor Systems. New York, USA, 2005: 255-267

[47] Shafi N B, Ali K, Hassanein H S. No-reboot and zero-flash over-the-air programming for wireless sensor networks//Proceedings of the 9th Annual IEEE Communications Society Conference on Sensor, Mesh and Ad Hoc Communications and Networks. Seoul, South Korea, 2012: 371-379

[48] Dong W, Liu Y, Chen C, et al. Elon: Enabling efficient and long-term reprogramming for wireless sensor networks. ACM Transactions on Embedded Computing Systems, 2014, 13(4): 77:1-77:27

[49] Gnawali O, Fonseca R, Jamieson K, et al. Collection tree protocol//Proceedings of the 7th ACM Conference on Embedded Networked Sensor Systems. California, USA, 2009: 1-14

[50] Tolle G, Culler D. Design of an application-cooperative management system for wireless sensor networks//Proceedings of the Second European Workshop on Wireless Sensor Networks. Istanbul, Turkey, 2005: 121-132

[51] Maróti M, Kusy B, Simon G, et al. The flooding time synchronization protocol//Proceedings of the 2nd International Conference on Embedded Networked Sensor Systems. Baltimore, USA, 2004: 39-49

[52] Pfammatter D, Giustiniano D, Lenders V. A software-defined sensor architecture for large-scale wideband spectrum monitoring//Proceedings of the 14th International Conference on Information Processing in Sensor Networks (IPSN). Seattle, USA, 2015: 71-82

[53] Yu M, Jose L, Miao R. Software defined traffic measurement with opensketch//Proceedings of the 10th USENIX Symposium on Networked Systems Design and Implementation (NSDI). Seattle, USA, 2013: 29-42

[54] Jeyakumar V, Alizadeh M, Geng Y, et al. Millions of little minions: Using packets for low latency network programming and visibility//Proceedings of the 2014 ACM Conference on SIGCOMM. Chicago, USA, 2014: 3-14

[55] Cao C, Luo L, Gao Y, et al. TinySDM: Software defined measurement in wireless sensor networks//Proceedings of the 15th International Conference on Information Processing in Sensor Networks(IPSN). Vienna, Austria, 2016: 1-12

[56] Fonseca R, Dutta P, Levis P, Stoica I. Quanto: Tracking energy in networked embedded systems//Proceedings of the 8th USENIX Conference on Operating Systems Design and Implementation (OSDI). San Diego, USA, 2008: 323-338

[57] Hui J W, Culler D. The dynamic behavior of a data dissemination protocol for network programming at scale//Proceedings of the 2nd International Conference on Embedded Networked Sensor Systems (SenSys). New York, USA, 2004: 81-94

[58] Panta R K, Khalil I, Bagchi S. Stream: Low overhead wireless reprogramming for sensor networks//Proceedings of the 26th IEEE International Conference on Computer Communications. Anchorage, USA, 2007: 928-936

[59] Dong W, Liu Y, Chen C, et al. R2: Incremental reprogramming using relocatable code in networked embedded systems. IEEE Transactions on Computers, 2013, 62(9): 1837-1849

[60] Qiu Jie-Fan, Qian Li-Ping, Huang Liang, Chen Qing-Zhang. Research on storage optimization of reprogramming applied in Internet of Things. Chinese Journal of Computers, 2016, 20(39): 1-15(in Chinese)
(邱杰凡，钱丽萍，黄亮，陈庆章. 面向物联网重编程的存储优化方法研究. 计算机学报，2016，20(39)：1-15)

DONG Wei, born in 1982, Ph. D., professor. His current research interests include Internet of Things and sensor networks, network measurement, wireless and mobile computing.

CHEN Gong-Long, born in 1992, Ph. D. candidate. His current research interests include wireless and mobile computing.

CAO Chen-Hong, born in 1991, Ph. D. candidate. Her current research interests include Internet of Things and measurement in sensor networks.

LUO Lu-Yao, born in 1992, M. S. candidate. His current research interests include Internet of Things and measurement in sensor networks.

GAO Yi, born in 1987, Ph. D., research assistant professor. His research interests include protocol design in sensor networks, measurement algorithm design, wireless and mobile computing.

Background

This work is supported by the National Natural Science Foundation of China (No. 61472360 and No. 61502417), the Zhejiang Provincial Key Research and Development Program (No. 2017C02044), and the CCF-Tencent Open Research Fund. Currently, SDSN research and development are still in the state of active development. There are lots of open problems that need further investigation to make SDSN more flexible and more practical. Providing a convenient programming model for software control is a continuing focus of current research. Design tradeoffs among energy efficiency, flexibility, and programming convenience, need to be made with respect to different application scenarios. While there is a large body of works devoted to wireless reprogramming and remote debugging, how to incorporate them into a unified SDSN framework is still an open problem. With a stronger support from a generic and efficient SDSN architecture, we envision that WSN application development and management will be greatly simplified.

In this article, we conduct a survey on the design of a software-defined architecture for embedded sensor networks. We present a novel taxonomy for software-defined sensor networks according to different abstractions of functionalities. We review recent progresses in this area. We also examine the major challenges towards a generic and efficient SDSN architecture. To address these challenges, we summarize some useful techniques. We believe a SDSN architecture can greatly facilitate software development for deployed networks, allowing rapid technical innovations.