A Survey on IP-based Wireless Sensor Networks Solutions

<table>
<thead>
<tr>
<th>Journal:</th>
<th><em>International Journal of Communication Systems</em></th>
</tr>
</thead>
<tbody>
<tr>
<td>Manuscript ID:</td>
<td>IJCS-09-0265.R1</td>
</tr>
<tr>
<td>Wiley - Manuscript type:</td>
<td>Research Article</td>
</tr>
<tr>
<td>Keywords:</td>
<td>Wireless Sensor Networks, Internet Protocol, IP-enabled Wireless Sensor Networks, Ubiquitous Computing</td>
</tr>
</tbody>
</table>
A Survey on IP-based Wireless Sensor Networks Solutions

Joel J. P. C. Rodrigues1,*, Paulo A. C. S. Neves2

1 Instituto de Telecomunicações, Portugal
Department of Informatics, University of Beira Interior, Covilhã, Portugal
email: joeljr@ieee.org

2 Instituto de Telecomunicações, Portugal
Department of Informatics, University of Beira Interior, Covilhã, Portugal
Polytechnic Institute of Castelo Branco, Castelo Branco, Portugal
email: pneves@co.it.pt

SUMMARY

Wireless sensor networks can be composed of thousands of smart sensing nodes that capture environment data for a sink node. Such networks present new challenges when compared to traditional computer networks, namely in terms of smart node hardware constraints, and very limited energy resources. Ubiquitous computing can benefit from wireless sensor networks, from the perspective that sensed data can be used on behalf of the user without explicit intervention, turning ubiquitous computing into a reality. Internet connectivity in wireless sensor networks is highly desirable, featuring sensing services at a global scale. Two main approaches can be considered: proxy-based or sensor node stack based. This second approach turns sensors into data producing hosts, also know as "The Internet of Things". For years the TCP/IP protocol suite was considered inappropriate for wireless sensor networks, mainly due to the inherent complexity and protocol overhead for such limited hardware. However, recent studies made connecting wireless sensor networks to the Internet possible, namely using sensor node stack based approaches, enabling integration into the future Internet. This paper surveys the current state-of-the-art on the connection of wireless sensor networks to the Internet, presents related achievements, and provides insights on how to develop Internet Protocol-based communication solutions for wireless sensor networks today.

Copyright © 2000 John Wiley & Sons, Ltd.

KEY WORDS: Wireless sensor networks, Internet Protocol, IP-enabled wireless sensor networks, ubiquitous computing, mote

1. Introduction

Wireless Sensor Networks (WSNs) were born in military applications like many consumer technologies in use today. These networks are composed of a potentially large number of smart sensing nodes, and one or more sink nodes or base stations. The smart sensing nodes have a
processing unit with memory, a wireless communication transceiver, one or more sensors and an autonomous power supply. Base stations collect smart sensor data, possibly presenting it to the user or relaying data to another system. The sink node is faced as a more powerful node, which allows more processing over the sensed data [1].

Compared to traditional computer networks, WSNs are based on small smart nodes with very limited processing power, small footprint, and especially limited autonomous power supply [2]. When a node’s power supply is exhausted, it looses capacity to transmit or receive information, disappearing from the network. As a result, network lifetime depends on node lifetime, which depends on node energy, resulting in a major difference from traditional computer networks.

The objective of a smart WSN node is to send sensor data to the sink node. However, since networks can be very large, the majority of nodes do not have enough transmitting power to reach the sink directly. To address this shortcoming, mesh networks and/or hierarchical routing is often employed. For years research was focused on finding new routing and transport protocols for WSN, namely with energy efficiency considerations. Dedicated routing protocols can be split into three categories - flat, hierarchical and location-based routing [3]. As for transport protocols they can be divided into congestion control and reliability guarantee [4], with the last one divided into upstream and downstream. The dedicated protocol research was conducted with the basis that new challenges need new solutions, and TCP/IP (Transmission Control Protocol/Internet Protocol) [5, 6] protocol suite was considered inappropriate for WSN.

Mark Weiser’s vision of computing back in 1991 demanded technologies that were not available at the time [7] - the ubiquitous computing vision. The vision that the computing world is much more than a desktop, where computers seamlessly interact with the user, augmenting reality, taking decisions on behalf of the user in a transparent and unobtrusive way. The user does not interacts with the machine, the machine interacts with the user in an efficient and suitable way. Ubiquitous computing needs sensing data to make informed decisions, creating a user context and enabling services to the user transparently. Sensed data can be the room temperature, the vacancy of a parking lot, or even the current health status of a user that must be transmitted to a hospital facility [8].

Data sharing is essential, since many systems may be interested on such data, on behalf of many users. For instance the parking lot available capacity, and vacant place location may be of interest to many users in the vicinity. Just this simple example may use different data sources such as current user location, available parking lots, parking lot vacancy, parking lot price, user expected occupancy time, among others. The Internet plays a major role on data dissemination, and although it does not reach every corner of the world, it is clearly the most widespread network, being itself pervasive in nature. As a result, WSN must be connected to the Internet, providing sensing services at a global scale [9]. WSN with Internet connectivity can make ubiquitous computing realistic.

The TCP/IP suite is the standard for Internet communications. The desired WSN Internet connectivity must use the TCP/IP suite at some point, be it through a gateway, bridge or router, enabling protocol transformation, or directly at the smart sensor level. However for years research was focused on the creation of special dedicated transport and routing protocols for WSNs, clearly stating that IP was not ready to tackle the many different challenges that sensor networks present. Issues like the limited hardware of the typical node (processing power and memory), header overhead, and the possible lack of global addressing schemes due to the
TCP/IP node stack issues led to proxy-based architectures to connect sensor networks to the Internet, where typically the sink node interfaces dedicated protocols to Internet TCP/IP and vice-versa. Challenges exist to be tackled and some researchers were not happy with this status quo. If TCP/IP works and is readily available, with many years of research, why not use it for WSNs as well? Moreover it served us well for several years, and still does now. Several proposals began to surface, from the tackling of the identified challenges like using header compression, or taking advantage of IPv6, like addressing space and auto-configuration, passing by real network prototypes, and commercial solutions. And today it is possible to deploy an IP-powered WSN. These proposals, challenges and proposals are surveyed in this work.

The remainder of the paper is organized as follows. Section 2 elaborates on the architectural perspective of WSN Internet connection, with an overview of the identified challenges that IP faces in WSNs when considering sensor stack based approaches. Section 3 presents related research efforts, providing a state-of-the-art analysis of IP over WSN. Section 4 points to software and hardware solutions to deploy IP-enabled WSN today, and finally Section 5 concludes the paper, pointing future research trends in this area.

2. Background

The Internet Protocol is a network routing protocol that aims to provide a connectionless packet transmission between hosts, being them on different networks and even with different network technologies. Current majority of implementations still use protocol version 4, with a relatively slow adaptation of version 6. Version 6 provides enlarged addressing space, clear address assignment rules, several built-in mechanisms (such as stateless auto-configuration), different and optimized header format, among other improvements. IPv6 is best known for its dramatically increased addressing space when compared to IPv4, than for its additional features. This section starts with the presentation of the two main architectures for WSN Internet connectivity: proxy-based and sensor node IP stack based. Then challenges for integration of IP at the sensor level are identified, namely at the implementation level. The challenges do not make sense if the proxy-based approach is used, since IP is not used internally on this approach.

2.1. Architectural perspective

Typical computer networks can be split into three major types: bus, star and tree topologies. On a WSN data from the smart nodes must reach the sink, in a single-hop, hierarchical, mesh or hybrid architecture. When considering the introduction of IP mainly to provide Internet connectivity, two major approaches exist: proxy based and sensor node stack based.

Figure 1 presents a proxy-based architecture. This architecture relies on the sink node to provide protocol bridging between the sensor network and the Internet, thus acting as a router, exposing sensing services to the outside. This kind of architecture has the advantage of relieving the nodes from the IP stack activities, with almost no disturbances to the network itself. As a result this architecture is very suited for sensor networks already deployed with dedicated protocols. However, it poses major burden on the sink, while also providing a single point of failure on the sink node.
Some designs consider the sink as a personal computer, however on real deployments cost and size issues are raised, rendering it inadequate for several deployment scenarios. Moreover, on some designs the sink operates autonomously without permanent connection to another network. On such scenarios, connection is achieved for data acquisition and maintenance sporadically. For instance, just a few seconds of contact by flying over a field, so the system must feature disconnected operation support.

Sensor node IP stack-based architectures use IP as the routing protocol to exchange data on the sensor network, while also enabling Internet connectivity. Two scenarios can be considered here, one that the sensor directly communicates with the Internet, and another where only the sink node(s) connect directly to the Internet. Although some research work focuses only on the sensor node stack, current implementations prefer the second approach, mainly due to energy considerations. The sink is used as a “router” between the WSN and the Internet, filtering unwanted traffic.

Sensing data is made available by the sink, while the sink itself may receive configuration and management data, sending to the WSN nodes as appropriate. This scenario, presented in Figure 2, uses TCP/IP stack for layer 3 and layer 4 communications inside the sensor network and through the sink, thus allowing Internet connectivity. Moreover, with the required adaptations management tools developed for IP may be used inside the network.

2.2. Challenges of IP over wireless sensor networks

Challenges were the main support for the idea that IP over WSN should be dropped against other approaches at the sensor level. These challenges can be identified as large header overhead for such a small packet wireless communication (e. g. the IEEE 802.15.4 standard [10]), the need for a global addressing scheme, the limited bandwidth (typically around 250Kbps or even lower), limited energy of the nodes, implementation challenges, and also the TCP transport protocol [11]. This section presents an overview of the major challenges for IP on WSN.
Large Header Overhead. IP packets like in many other routing protocols are divided into two parts: message header and body (the payload). The header is faced as overhead for the communication, since the payload is on the message body itself. On a smart sensor node, the hardware that consumes more power per working unit (one can consider a working unit as the execution of an instruction on the processor, or a bit transmitted on the wireless transceiver for instance) is the wireless communication module. As a result, more bits being transmitted or received greatly affects energy consumption, even if the transceiver is just idle listening. IPv4 header has a minimum of 20 bytes on the header, while IPv6 goes to 40 bytes, mainly due to destination and source address that occupy 128 bits each. In order to tackle this challenge a header compression mechanism must be used.

Global Addressing Scheme. IP relies on a global addressing scheme with unique addresses, both source and destination must have an IP address in order to communicate. IPv4 may use Dynamic Host Configuration Protocol (DHCP) for address auto-configuration, although that poses overhead, with more traffic just for configuration purposes. However with IPv6 one can apply stateless auto-configuration, ditching the need for a dedicated DHCP server. On a wireless sensor network the data is more important than the address of the node it comes from. This WSN characteristic is referred as data-centric routing, as opposed to address centric routing employed on computer IP networks. Moreover, many networks employ some sort of data aggregation, so data does not come from a specific node, but from some processing over a group of node’s data.

Limited Bandwidth. Typically WSNs nodes communicate at speeds of 250 Kbps on IEEE 802.15.4 radios, or even less. With such a limited bandwidth, the transmission of more bits per packet increases medium access delay and transmission time, which can be critical in some scenarios. As a result, protocols must ensure that overhead is kept to a minimum, only transmitting the needed information. Also, broadcasts must be avoided in the network, or at least minimized. Clearly TCP/IP must be optimized for energy efficiency.

Limited node energy. Wireless sensor network nodes use energy from a small internal battery, and in many scenarios recharge/replacement is just not an option, even more if the network has many nodes and/or is placed on hostile environments. Wireless communication draws energy faster than any other smart sensor node hardware component, so transmission and reception must be kept to a minimum, using smart duty cycle mechanisms when possible. In some cases the cost of a bit transmission equals the energy of executing 100 instructions. When a given number of nodes stop working due to energy depletion, the network is no longer a network, sensing services may be stopped, thus rendering the network useless.

Implementation Challenges. IP presents several implementation challenges on the limited node hardware. A big implementation challenge is the memory that IP needs for full operation, even more if considering packet reassembly due to fragmentation. Moreover, MAC layer protocols are designed for small packet size, making it hard to accommodate typical IP maximum transmission unit (MTU). The IP MTU on a wire network can easily achieve 1500 bytes, while the maximum physical layer packet size is 127 bytes for IEEE 802.15.4 networks. On IPv6 for instance the MTU is typically 1280 bytes. Another challenge is the processing power needed for such a relatively complex protocol, moreover if routing capabilities must also be incorporated.

Transport Protocol. IP is a best-effort routing protocol, does not guarantee reliability on packet delivery. The obvious candidate to achieve reliability is the TCP transport protocol. However, TCP was not designed for the burst error rate of wireless transmission on a sensor
network, assuming that a packet loss is always due to congestion. Moreover it does not tackle energy considerations, and poses significant overhead with its end-to-end acknowledgment mechanism. Nevertheless not as robust as TCP, User Datagram Protocol (UDP) may also be considered, especially for applications where the loss of some packets is not critical. Moreover, support for TCP and/or UDP may allow WSN nodes to communicate seamlessly with existing IP devices (namely computer hosts).

3. Related research efforts

This section elaborates on a plethora of published work related to IP over WSN. From the development of a dedicated IP stack for low processing power microprocessors, to real deployments, passing through some prototypes and IP sensor stacks, this paper surveys the research work that made it all possible.

The first breakthrough on the implementation of a TCP/IP stack for smart sensor nodes came from Adam Dunkels when he developed uIP and lwIP TCP/IP stacks [12]. While uIP was primarily developed for very low processing power systems with 8-bit processor architectures, lwIP presents a larger footprint for more capable systems. Both solutions conform to a subset of RFC 1122 [6] and feature IP, TCP and ICMP protocol implementations. lwIP also supports more than one network interface per device, with UDP support. This achievement allowed the demystification that TCP/IP was too complex for WSNs smart nodes, which typically feature 8 or 16-bit processor architectures.

An example of an IP-based WSN implementation for intrusion monitoring is presented in [13]. The network employs ESB platform from FU Berlin motes, running the Contiki operating system and uIP stack. For node address configuration the authors used spatial IP address assignment, which use node position coordinates. An overlay network is used, where a set of “core” nodes form a backbone, receiving alarm events from the smart sensor nodes. The backbone network replicates and transports events among its nodes, featuring Network News Transfer Protocol (NTTP) for external connectivity.

A Body Sensor Network (BSN), a network where sensor nodes capture human health parameters in an unobtrusive way can also benefit from IP-enabled motes [14]. Researchers at HP labs developed an IP-enabled BSN featuring TCP/IP for motion data capture [15]. The nodes use TinyOS operating system and uIP stack over IEEE 802.15.4, featuring a small amount of flash memory to store data, enabling disconnected continuous operation.

An IP-enabled wireless sensor network architecture named IPSense, that features flexible addressing, enhanced mobility (an issue many times ignored, but very important at different levels), demonstrating the fall of many myths related to the use of IP over WSN is presented in [16]. IPSense explores node aggregation through the use of cluster heads designated as Sensor Routers that manage communication to the sink node. Sensor Routers are faced as gateways between the sensor network and other networks, and benefit from more hardware resources when compared to smart sensor nodes.

IPv6 adds a larger address space, however it may introduce a greater overhead when compared to IPv4, due to its fixed 40 bytes header with 128-bit addresses. However authors of [17] present a study comparing IPv4 to IPv6 for WSNs, based on simulation and a network prototype running ContikiOS. They prove that IPv6 has low impact and is a perfectly feasible solution. Moreover, in the conclusions they point some of the advantages of integrating IPv6 on
WSNs, like stateless auto-configuration, enlarged addressing space to cope with large networks, and the growing utilization of IPv6 all over the world.

The 6LoWPAN working group (6LoWPAN WG) from the Internet Engineering Task Force aims at the development of an intermediate layer that can map IPv6 protocol functionality over IEEE 802.15.4 [18]. IEEE802.15.4 is a wireless communication standard that defines data-link and physical layers, used on the majority of sensor hardware and also on ZigBee [19]. In terms of frames, two main issues arise when using IPv6 over IEEE802.15.4: header overhead of IPv6 and low payload of IEEE802.15.4 (only 102 bytes). For the first issue 6LoWPAN uses a compression mechanism (designated HC1, HC01 is in draft stage), and for the second a fragmentation one. Fragmentation support can split one IPv6 datagram into several IEEE 802.15.4 frames, providing an adaptation layer between IPv6 and IEEE 802.15.4 low power devices, while reducing IPv6 overhead inside the sensor network.

Another very interesting contribution comes from the author of the world’s first complete IPv6/6LoWPAN stack for low-power wireless networks and his advisor [20]. In the paper several considerations on the use of IPv6 over WSN are made, together with some results on a real scenario. The authors provide valuable information on the three fundamental services of an IP network - configuration and management, forwarding, and routing. The work also includes evaluation of the proposed IPv6 architecture for WSNs with TelosB motes from Crossbow. The implementation with one UDP socket and a TCP connection consumes around 23.5KB of program memory and 3.5KB of RAM. The results demonstrate that the architecture implementation efficiency is greater than on WSN without particular architecture.

The evolution of uIP (IP version 4) is uIPv6 [21]. This software layer provides applications in the Contiki Operating System with a TCP/IP stack, and maps IP datagram on IEEE 802.15.4, 802.11 and Ethernet. uIPv6 uses 6LoWPAN for IEEE 802.15.4 MAC and link layer, and RFC 2464 (Ethernet adaptation layer) for IEEE 802.11 and Ethernet. The stack presents TCP, UDP, IPv6 Addressing, ICMPv6 and Neighborhood Discovery (ND), requiring less than 2KB of RAM and 11.5KB of program memory. ContikiOS features this stack since version 2.2.3, and can be used today on several of the hardware platforms covered. For a complete solution on Atmel RAVEN platform the code size is 35KB, which includes RAVEN drivers, IEEE 802.15.4 MAC and Physical layer implementation, 6LoWPAN with fragmentation and header compression, uIPv6 and the ContikiOS. The UDP transport layer adds 1.3KB, while TCP adds 4KB of code size.

From the analysis and evaluation of inter and intra network communication with 6LoWPAN, a relatively high overhead when data flows between different networks can be identified. This is mainly due to bits dedicated to addressing, so 6GLAD architecture presents a twice-NAT, and reverse network address translation [22]. Twice-NAT provides both source and destination IP addresses modification. By using a reverse NAT mapping on the WSN gateway, IPv6 and local link addresses are mapped, allowing the use of short addresses for low power nodes, while enabling outside IPv6 hosts to reach inside nodes. The proposal is validated through simulation.

The IEEE 802.15.4 standard can use two types of addresses: 16-bit short addresses and 64-bit EUI-64 addresses. In [23] authors present a Dual Addressing Scheme (DAS) for IPv6 over IEEE 802.15.4 networks. DAS combines a global unicast address and a link local address to save energy and resources. The gateway is responsible for transforming local-link smaller addresses to full IPv6 addresses and vice-versa, using a translation table on the gateway. With this mechanism nodes have unique global addresses, but the burden of such addressing sticks
with the more powerful gateway. The paper also presents some simulation results.

Providing address inter-networking between WSN and Internet, while supporting web services is presented in [24]. The authors present and validate a solution for ZigBee-based nodes and Internet based on TinyOS. The implementation uses Mica motes and dual protocol stack at the sink level - ZigBee and IP. The sensor IP address can be assigned statically or dynamically and the client running a web browser can access the sensor through its IP address. The approach used a personal computer with Java and JSP technologies, and assigned an IP address to the mote, associating it with the node’s ZigBee ID. However this approach relies on the gateway to transform ZigBee on the WSN to TCP/IP on the Internet.

A new architecture that extends tree-based routing algorithm in IPv6 over WSN, is the Extended Tree-based Routing Algorithm (ETRA), presented in [25]. Authors describe a ZigBee hierarchical routing algorithm with distributed address assignment mechanisms, and propose an extension featuring three steps - descendant search, nearest neighbor search and ancestor search. The paper also presents a performance evaluation study. By using the information about neighboring nodes, ETRA can improve on the hierarchical routing methods.

The LoWPAN Network Management Protocol (LNMP) management architecture provides 6LoWPAN network management [26]. Authors present the need for management architecture because 6LoWPAN networks are not just ordinary IPv6 networks, since nodes are very constrained in terms of hardware resources. The SNMP (Simple Management Network Protocol) protocol is translated to and from a simplified format on the 6LoWPAN gateway, state information is kept on coordinators, which are responsible for all its subordinate devices. The gateway is responsible for state information from the list of reporting coordinators. Authors conclude that the proposed management architecture effectively reduces communication cost, while providing support for the SNMP protocol.

A configurable tiny TCP/IP protocol stack featuring SIP (Session Initiation Protocol) for sensor networks is presented in [27]. The protocol stack was implemented on very constrained devices with 512 bytes or 1KB of system RAM, based on Atmel 8-bit microprocessors. For IP address assignment the authors use spatial information, which may lead to duplicated addresses. At the network layer IP and ICMP are supported, while TCP compression and UDP are supported at the transport level. A reduced version of SIP is supported on top of UDP. However no more information about interoperability with other platforms, namely the compatibility with existing operating systems, is given.

A gateway for IP Multimedia Subsystem and WSN interworking, brings together two very interesting research areas [28]. Authors consider IMS - IP Multimedia Subsystem as the de-facto standard for IP-based multimedia services and present a gateway to WSN, opening a new world of multimedia services. Validation is performed using Ericsson IMS simulated environment. Although very interesting, the work does not tackle WSN challenges nor it takes into consideration WSN protocol issues.

A light TCP/IP stack for small wireless nodes, the Compact Wireless-TCP/IP (CW-TCP/IP), can be found in [29]. The stack adopts TCP Veno as a congestion control algorithm and does not references external data structures. The experimental environment includes comparison with other two approaches: Linux TCP/IP and µC/TCP-IP of µC/OS-II. The proposed stack features lower code size, higher average bandwidth and lower CPU utilization. However the hardware is not common, namely a 233MHz AMD Geode SC1100 processor with 256MB of RAM, too powerful for a smart sensor node to use, leading to higher energy consumption, inadequate size and cost.

Copyright © 2000 John Wiley & Sons, Ltd.

Prepared using dacauth.cls

http://mc.manuscriptcentral.com/iwcs

Int. J. Commun. Syst. 2000; 00:1–6
Indeed ubiquitous computing can benefit from IP-based sensor networks, namely in healthcare monitoring [30]. The authors propose a prototype of a global home care monitoring, allowing patient mobility. The patient uses a body sensor network with several biosensors that communicate with a single 6LoWPAN enabled node. This local node performs data routing between biosensors, connecting wirelessly with the gateway. The gateway is connected to the Internet, enabling medical data long-term storage and further analysis. The approach is validated through simulation on the Octopus simulator.

Authors of [31] describe a gateway-based framework for sensor network/IP network interconnection. The approach identifies challenges that a sensor node IP stack faces and conclude that gateway-based approach is best suited for IP interconnection. The framework defines three entities - the sensor node, the IP host and the gateway node - and two main components - virtual address assignment and packet translation. The gateway performs a two mapping process by assigning IP virtual addresses to sensor nodes. The gateway maps WSN internal IP virtual addresses into IP global addresses and vice versa.

An architectural design of lightweight IPv6 for low-power wireless personal area networks, LWIPv6, is proposed in [32]. LWIPv6 presents a basic protocol functionality, removing routing functions, featuring host-only functionality, does not supports multiple interfaces, and presents no multicast, Mobile IPv6 or IPSec support. The design uses hierarchical protocol configuration, namely IEEE 802.15.4 and multihop routing in layers 1 and 2, 6LoWPAN, IPv6 and ICMPv6 at the network layer and finally UDP at the transport layer, lacking TCP.

Support for IPv6 label switching on IEEE 802.15.4, with the description of an "intuitive approach" for IPv6 packet transport on ZigBee-based wireless personal area networks (WPANs) is described in [33]. The work is based on 6LoWPAN implementation over TinyOS to support routing and management capabilities, taking advantage of the mesh option. As a result the paper concludes that it is possible to extend point-to-point WPAN connections to multipoint-to-multipoint approach, integrating suitable on-demand routing mechanisms.

IP-based WSN can also be used on disaster management scenarios [34]. Authors propose an architecture for the sensor node and gateway as presented in Figures 3(a) and 3(b) respectively. The sensor node protocol stack uses IEEE 802.15.4 on its MAC and Physical layers and an adaptation layer through 6LoWPAN specification. Besides using header compression and the transport specification of 6LoWPAN, the authors also use ad-hoc routing at link layer, adding LOAD to the 6LoWPAN layer, thus providing a lightweight multi-hop routing solution. The architecture features Neighbor Discovery Protocol (NDP) and Stateless Auto Configuration (SAA), with modifications to limit the bandwidth consumption. On the gateway, Internet connection is guaranteed through a series of interfaces - Wireless Local Area Network (WLAN), General Packet Radio System (GPRS), and even Satellite Link - that the gateway chooses on demand. The gateway has the same 6LoWPAN layer and IEEE 802.15.4 network adapter as the nodes, performing transformation at the IPv6 layer, allowing connectivity to the global Internet.

ZigBee presents two solutions for Internet connectivity: gateway and bridge [35]. The gateway offers a full featured connectivity, while allowing a greater diversity of applications and devices interconnected by ZigBee networks. Gateways convert wireless protocols and sensor data into various formats, for instance Ethernet. Gateways allow WSNs to be integrated into existing applications, such as SCADA or Modbus in industrial environments. The ZigBee Bridge [36], presented in Figure 4, presents a simpler approach than gateways, but application space is smaller. This new architecture performs packet switching at layer 2 (data-link layer),...
enabling the use of the bridge without the help of a user-defined application.

The ZigBee Bridge can act as a pure "router" node between the ZigBee WSN and the IP network. The bridge has three main components: the Bridge Routing Layer (BRL), the Bridge Device Layer (BDL) and the ZigBee IP Transport (ZIPT). The BRL is responsible for packet dispatch to the appropriate interface (native IEEE 802.15.4 or IP). The BDL encapsulates/de-capsulates message formation, and assures connection maintenance. It also provides data and management services (including transmit and receive packets via IP connection). Finally the ZIPT is responsible for packet transportation over IP networks, including ZigBee frames and Bridge Device control frames, such as connection request [37].

The presented research works for IP over WSN clearly state that WSN can feature Internet connectivity. Moreover, both approaches are present - proxy based and sensor node based. The 6LoWPAN specification drives several implementations, while other prefer to develop their own sensor stack.

4. Current available solutions

This section presents information on current available resources and solutions to build an IP-based wireless sensor network. Basically, the developer needs to choose a suitable operating system and hardware platform. Although some approaches feature only the TCP/IP stack without specific operating system, this section elaborates on software as an operating system.

The developer's choice is not trivial, since not all combinations of hardware/software are possible. Since hardware solutions are clearly in more number than software, this section starts with a view on the two most popular software solutions: ContikiOS and TinyOS. Some hardware platforms are supported by both operating systems, giving some degree of freedom to the developer. Another approach is to add a new hardware port to an existing operating system, but such task may turn out to be complicated and time consuming.

4.1. Software

Implementations of node software for IP can rely on two of the major operating systems available for embedded systems: TinyOS [38] and ContikiOS [39, 40]. TinyOS was developed and is maintained by University of Berkeley, CA, and supports the large majority of sensor

![Figure 3. Sensor protocol stack and gateway according to [34].](image-url)
nodes manufactured by Crossbow Technology, such as TelosB, IRIS, Mica (version z and 2), Imote2, among others. The Shimmer platform is also supported, although users experience some problems namely with the Bluetooth radio driver and microSD interface. TinyOS is currently on version 2.1, benefiting from a large developer base that contributes for this operating system. A virtual machine running XUbuntu can be obtained form the Website and very recently a book on TinyOS programming was released [41].

TinyOS programming language is nesC (networked embedded systems C), a language with syntax close do ANSI C, but with notable differences in the programming model, imposing a certain learning curve on new adopters. The layered operating system presents a component-based architecture, with components and interfaces. A component provides a set of services, while several components connected provide features to the application. TinyOS web page has all the information regarding download, a wiki for documentation where the user can download the valuable e-book from Philip Levis “TinyOS programming Manual”, getting started tutorials (valuable resources to start programming) and download information.

TinyOS also features a 6LoWPAN implementation, blip - Berkeley IP Information, which draws on a previous implementation - b6lowpan. The stack features IPv6 neighbor discovery, default route selection and point-to-point routing. It also supports ICMP, UDP and includes a prototype TCP stack [42].

ContikiOS is developed and maintained by a core group of researchers led by Adam Dunkels from the Swedish Institute of Computer Science, and currently supports ESB platform, Tmote Sky (with the respective evolution - TelosB), Atmel AVR Raven platform, Sentilla, Micaz and Sensinode (very recent ports - 9th September 2009), among others. ContikiOS was awarded the “IPv6 ready” silver seal (according to [39]), and is currently on version 2.3, released 27 June 2009. ContikiOS can be downloaded on the website, through a virtual machine implementation of Ubuntu 8.04 fully configured for Contiki development, or just the Contiki system itself.

IPv6 implementation is based on a new version of uIP, the uIP6 stack, which demands 11,488 bytes of program memory and 1,748 bytes of RAM. uIP6 presents the same programming interface as uIP, and is tightly coupled with both UDP and TCP. ContikiOS also features an implementation of RFC 4944, regarding 6LoWPAN working group efforts. ContikiOS website also has all information regarding download, documentation, tutorials and latest news.

ContikiOS uses standard C language for development, and provides a virtual machine image.
based on Ubuntu 8.04 LTS, the instant-contiki. Instant-contiki was also updated with the release of Contiki 2.3 version, turning it into an approximately 1GB download. Tutorials are also available for running this virtual machine on Mac OS X compatible virtualization software. This constitutes the easiest way to get Contiki running, specially when using Microsoft Windows or Mac OS X operating systems.

ContikiOS provides a event-driven kernel, where events can be defined by the system or the developer. A threading model is provided through protothreads, a mechanism that allows multi-threading with a very efficient memory allocation approach - the stack is not dedicated to a single thread, but is reused among the defined protothreads. The operating system also features event timers, two main communications stacks - uIP and Rime, and a system architecture that allows recompilation for several platforms without code changes.

Both TinyOS and ContikiOS present a simulation tool, respectively TOSSIM and COOJA. TinyOS simulator, TOSSIM, enables mote simulation to ease the development cycle, simulating at the bit level [43]. TOSSIM has a GUI, TinyViz, which can both visualize and interact with simulations. The simulator tries to simulate TinyOS and its execution, instead of the real world. For instance it does not execute time, so it may be difficult to asset performance based on simulation results provided by TOSSIM.

COOJA is a Java-based simulator developed for ContikiOS [44]. It allows the simulation/emulation of mote behavior for several mote types, allowing creation of different firmwares based on ContikiOS code. A suitable GUI is also provided, with full graphical interface and console output.

On a side note about simulation, another simulated environment, MSPsim is provided by researchers of the SICS institute, the same as ContikiOS, and is as such part of ContikiOS [45]. It can emulate the behavior of a single MSP430-based mote, namely the sky/TelosB and ESB platforms. MSPsim is made in Java, presents a GUI and simulates the behavior of the firmware at the instruction level. This simulator/emulator is very useful for debugging stand-alone code.

Nano-RK [46] is another another approach that is less known than the previous two, but also worth mentioning [47]. Nano is a real-time operating system featuring multi-hop networking support for WSNs. Created at the Carnegie-Mellon University it is still young and only supports the MicaZ and FireFly hardware platforms. It uses around 16KB of program memory and 2KB of RAM. This real-time kernel features a C GNU toolchain, built-in fault handling, energy-efficient scheduling, among other interesting features. However, to the best of our knowledge this kernel does not feature IP support at the time of writing, but can use an emerging standard, SLIPIstream, for IP sensor node gateway communication. Nevertheless, IP is not used on the WSN nodes.

4.2. Hardware

The majority of hardware solutions are based on two microprocessor groups: one from Texas Instruments with the MSP430 family and another also very popular from Atmel semiconductor, the ATmega. In terms of radio, the standard IEEE 802.15.4 compliant is the most common, with a 250kbps maximum transfer rate, very common is the Texas Instruments CC2420/2430 transceiver, also known as Chipcon CC2420 [48]. Crossbow technology [49] manufactures and sells several wireless sensor nodes (motes): IRIS, MICAz, MICA2, Imote2, TelosB and Cricket. Other motes can be purchased through Scatterweb [50], a spinoff company from FU Berlin, producing among others the ESB platform (although recently to be discontinued);
A SURVEY ON IP-BASED WIRELESS SENSOR NETWORKS SOLUTIONS

Table I. Some examples of wireless sensor hardware (motes).

<table>
<thead>
<tr>
<th>Mote</th>
<th>Processor Platform</th>
<th>RAM/program /measure</th>
<th>Transceiver</th>
<th>Operating Systems</th>
<th>Price (USD)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Crossbow Iris</td>
<td>Atmel ATmega 1281</td>
<td>8KB/128KB/512KB</td>
<td>Atmel RF230</td>
<td>TinyOS</td>
<td>115/75</td>
</tr>
<tr>
<td>Crossbow MICAz</td>
<td>Atmel ATmega 128L</td>
<td>4KB/128KB/512KB</td>
<td>TI CC1000</td>
<td>TinyOS</td>
<td>99/75</td>
</tr>
<tr>
<td>Crossbow MICA2</td>
<td>Atmel ATmega 128L</td>
<td>4KB/128KB/512KB</td>
<td>TI CC1000</td>
<td>TinyOS/nano-RK</td>
<td>115-125/75</td>
</tr>
<tr>
<td>Crossbow Imote2</td>
<td>Marvel PXA271</td>
<td>32MB/32MB/1MB</td>
<td>TI CC2420</td>
<td>TinyOS</td>
<td>299/190</td>
</tr>
<tr>
<td>Crossbow TelosB</td>
<td>TI MSP430</td>
<td>10KB/48KB/1MB</td>
<td>TI CC2420</td>
<td>TinyOS/ContikiOS</td>
<td>99-139</td>
</tr>
<tr>
<td>Atmel AVR Raven</td>
<td>Atmel ATmega 1284P</td>
<td>5KB/65KB/1MB</td>
<td>19.2Kbps</td>
<td>ContikiOS/Proprietary</td>
<td>85 (kit)</td>
</tr>
<tr>
<td>Scatternode centering</td>
<td>TI MSP430</td>
<td>5KB/65KB/1MB</td>
<td>19.2Kbps</td>
<td>Proprietary/ContikiOS</td>
<td></td>
</tr>
<tr>
<td>Shimmer</td>
<td>TI MSP430</td>
<td>10KB/48KB/2GB</td>
<td>TI CC2420</td>
<td>TinyOS/x.x</td>
<td>220</td>
</tr>
<tr>
<td>Coalesences iSense</td>
<td>Jennic JN5139</td>
<td>5KB/128KB/1MB</td>
<td>Integrated</td>
<td>Proprietary</td>
<td></td>
</tr>
<tr>
<td>Sun SPOT</td>
<td>ARM926T</td>
<td>4KB/4KB/1MB</td>
<td>IEEE 802.15.4 radio</td>
<td>Java Programming</td>
<td>299 (edu. kit)</td>
</tr>
</tbody>
</table>

Note: The table summarizes some of the currently available hardware platform’s characteristics, in terms of processor platform, memory (RAM, program memory, and external flash for sensing data storage), transceiver chip type, and supported operating systems.

From the analysis of Table I, typical mote RAM ranges from 4KB on MICA2, MICAz, and Cricket (this mote is a MICA2 version with ultrasound sensors) up to about 10KB. As expected, all nodes have program memory, ranging from 48KB to 128KB. The Imote2 is the most powerful of the set, with a PDA-like processor (PXA architecture), 32MB of RAM and 32MB of program memory, making it a good candidate for sink node implementation. However, it is also the most expensive, the biggest in size, needs more batteries, and is the most power drain than other less costly approaches. Moreover, the Imote2 board does not have any sensors or programming interfaces built-in, so the developer must buy a separate sensor board (ITS400), and the corresponding USB programming board (IIB2400). Platform costs range from 150USD for the interface board and 249USD for the sensor board. Another version of this mote, the Imote2.NET, runs the Microsoft .NET Micro Framework.

Some motes feature a dedicated flash memory to store sensor data, allowing disconnected operation without sacrificing program memory. In this regard, the Shimmer platform boosts a “huge” 2GB of storage, through the use of a microSD card (non-SDHC). This solution clearly enhances connectivity, since the card can be extracted and by the means of an adapter be plugged into a laptop for convenient data retrieval and latter analysis.

Copyright © 2000 John Wiley & Sons, Ltd.

Prepared using dacauth.cls

International Journal of Communication Systems

http://mc.manuscriptcentral.com/ijcs
At the time of writing author’s research is being performed on Crossbow TelosB where, for 129USD one gets, three onboard sensors (humidity, temperature and ambient light), an MSP430 processor with 10KB RAM and 48KB of flash programming storage, CC2420 IEEE 802.15.4 radio and USB programming. It can be considered a all-in-one developing solution, even integrating a two AA battery slot for autonomous operation. Another TelosB version exists, where for 99USD the developer gets the same hardware, with the exception of the built-in sensors. Another advantage of this hardware platform is the ability to develop on both ContikiOS and TinyOS operating systems.

As for power supply, Crossbow opted for two AA kind batteries contributing heavily to size and weight (exception is Imote2 with 3 AAA), while for instance Shimmer opts for a dedicated Li-ion 230mAh battery that can be recharged through a dedicated charger or via USB port. However, for academic research, this also pinpoints to a more wearable nature of the Shimmer platform, featuring size and weight characteristics that are body sensor network friendly. Crossbow technology has an IRIS OEM single chip module for integration on other hardware designs, but power supply and sensors must be provided separately.

A commercial version from Arch Rock, the PhyNet™ product suite, uses 6LoWPAN for wireless sensor networks, by integrating an IP stack on every sensor node [55]. The system architecture is divided into the three following layers: the sensor node, the PhyNet™ router and the PhyNet™ Server. There are two types of nodes available, the IPserial node supports IEEE 802.15.4 communication to PhyNet™ routers, a RJ-45 connector that supports RS-232 and RS-485, a wide variety of sensors, but needs an external power supply of 5V. The other node, the IPsensor node, features an internal battery, integrated sensors for temperature, humidity, photo synthetically active radiation and total solar radiation, connection to PhyNet™ routers and a USB 2.0 port.

The PhyNet™ router connects 6LoWPAN wireless networks to Ethernet and Wi-Fi networks, providing Internet connectivity to wireless sensor networks. One wireless sensor network may have multiple routers, enhancing global connectivity, enabling dynamic routing around failures or radio frequency interference. Finally the PhyNet™ server can manage multiple wireless sensor networks, providing a uniform and common vision through web services architecture and web browser interface. The server can translate embedded sensor applications to web services, and let users setup, monitor and manage multiple WSNs.

Another company that also supplies hardware for sensor networks, namely with integrated designs and 6LoWPAN implementation is Jennic [56]. The Jennic JN5139 microcontroller features a system-on-a-chip (SoC) design with integrated RAM (8KB to 96KB), integrated ROM (192KB), IEEE 802.15.4 compliant radio. The company also provides 6LoWPAN stack and software development kit. However, to the best of author’s knowledge, no full port for this platform exists for TinyOS or ContikiOS.

Recently, Atmel also provides on its web page a complete IEEE 802.15.4-compliant, IPv6/6LoWPAN based and ZigBee™ certified wireless solutions based on its RF transceivers, AVR and ARM microcontrollers, featuring free software stacks. Also ContikiOS and Arch Rock IP/6LoWPAN software distributions are available for the AVR Raven platform.

TinyOS, ContikiOS, dedicated commercial versions. Which is better? If looking for a rapid solution with minimal deployment time go commercial. If budget is tight and time is not critical both TinyOS and ContikiOS provide a software platform and extensive supported hardware base to tackle almost any application scenario. Being both open source, the developer has freedom to configure and program the solution.
Both operating systems feature 6LoWPAN implementations, adequate tools and examples, and good documentation. However, we believe that ContikiOS is one step ahead of TinyOS when considering IP-related approaches, namely uIPv6 is a solid implementation when compared to blip. At the developer level TinyOS uses nesC against the more familiar C of ContikiOS. However, clearly that is not a decisive issue. For testing purposes on real hardware the TelosB, an all-in-one platform, supporting both operating systems, is a good candidate for operating system evaluation.

5. Conclusions

This paper presented an overview of the challenges, architectural overview, main contributions, and resources to enable IP on wireless sensor networks, namely solutions that feature Internet connectivity. Integration of IP over wireless sensor networks still raises many eyebrows, especially with potentially large sensor networks with thousands of nodes sparingly distributed. Since large networks will require multi-hop routing, IPv6 is believed to be more resource-consuming than other specific approaches. However solutions like hierarchical routing or even employing several sink nodes on such large networks can clearly minimize energy consumption impact.

We also presented an overview of the major hardware platforms for sensor nodes and operating systems. The choice of a suitable research software/hardware combo is more a matter of taste than it is related to features. IEEE 802.15.4. is a de-facto standard at the MAC and physical layers for smart embedded wireless devices. While ContikiOS opted for a ANSI C approach with strong features like the coffee file system, protothreads and event timers, TinyOS opted for nesC. TinyOS supports a larger number of hardware platforms, however ContikiOS is catching up and presents the IPv6 Ready silver seal. 6LoWPAN itself will evolve, and TCP as we now it today may have to be modified to suit the challenges of wireless networks.

The two main approaches for Internet connectivity, namely proxy-based and sensor stack based, will merge into one single solution. The nodes will have a small set of IPv6-related protocols and features, namely with auto-configuration, while still acting as reduced function devices. Routing and other more processing power demanding tasks will be delegated to the sink, that will feature Internet connectivity. Routing at the IPv6 layer in such energy-constrained sensors is a major concern that may prevent it from happening. Smart nodes will act like hosts, while sinks will act like routers in an analogy with computer networks. IPv6 adoption is still slow, mainly because current IP version is still achieving satisfactory results. However as new devices are integrated, more services are required, and Internet will spread across the five continents, we will need IPv6.

The future is ours to shape, and it will be IP-based. Wireless sensor networks are still trying to find the "killer application" that will mainstream its use. Nodes are still not cheap as envisaged some years ago, mainly promised by MEMS (Micro Electro Mechanical Systems) technology. Albeit some commercial solutions, they are still applied in specific scenarios. Ubiquitous computing promised the invisible computer, almost a genie-in-a-bottle that can guess our intentions and desires, reacting accordingly. We still do not have this vision, but IP-based sensor networks may help in this future trend.

Wireless sensor networks will be integrated in the Internet of the future, namely to power
ubiquitous computing, providing sensing services. From environmental monitoring to more user-centric applications such as the parking lot example, there is a plethora of applications that can benefit. The Internet of Computers is long overdue, "IP is dead, long live IP for wireless sensor networks", and the Future Internet must take into account small devices that do not provide full web pages, but sometimes-critical information about human kind. This is how we see Internet in the near future. The "Internet of Things" is a reality!

Nevertheless much has to be done to achieve the desired ease-of-use of the current approaches, namely off-the-shelf ones. The integration of a deployment tool with simulation that can automatically program and define deployment scenarios for IP-based wireless sensor networks is a very challenging task. User adoption depends greatly on ease-of-use, perceived features, and cost/benefit ratio. A Plug-and-Play operation of sensor networks is mandatory to enable further adoption: just plug and use new sensors and new networks, no configuration needed. From the specification of a suitable description of each mote type, passing by the discovery of new nodes, service auto-configuration to a zero user configuration network deployment. Because a user does not need to know how a car is made to drive it, it should not be worried about implementation details of wireless sensor networks to get sensing services.

ACKNOWLEDGEMENTS

Part of this work has been supported by Instituto de Telecomunicações, Next Generation Networks and Applications Group (NetGNA), Portugal, and by the Euro-NF Network of Excellence from the Seventh Framework Programme of EU.

REFERENCES

A SURVEY ON IP-BASED WIRELESS SENSOR NETWORKS SOLUTIONS


Prepared using dacauth.cls