Sensor OpenFlow: Enabling Software-Defined Wireless Sensor Networks

Tie Luo, Hwee-Pink Tan, and Tony Q. S. Quek

***Abstract*—While it has been a belief for over a decade that wireless sensor networks (WSN) are application-specific, we argue that it can lead to resource underutilization and counter-productivity. We also identify two other main problems with WSN: rigidity to policy changes and difficulty to manage. In this paper, we take a radical, yet backward and peer compatible, approach to tackle these problems inherent to WSN. We propose a Software-Defined WSN architecture and address key technical challenges for its core component, Sensor OpenFlow. This work represents the first effort that synergizes software-defined networking and WSN.**

***Index Terms*—Software-defined networking, OpenFlow, wireless sensor networks.**

I. Motivation

EVER since their debut over a decade ago, wireless sensor networks (WSN[1]) have been conceived to be application-specific [1], which has probably formed a belief thus far. However, we deem it worth an afterthought, with the upsurge of sensor applications nowadays and the advancement of sensor technologies such as multi-modal and high-end mote platforms (e.g., Imote2, Stargate and NetBridge). These new changes render application-specific WSN prone to (1) *resource underutilization*, where multiple WSN for respective applications are deployed in the same or overlapping terrain while a single "versatile" WSN could achieve the same objectives, and (2) *counter-productivity*, where different vendors develop WSN in isolation without adequately *reusing* common functionalities which would considerably expedite prototyping and production.

Another problem with WSN is that they are rigid to policy changes. Policies are rules related to network-exogenous factors such as business operation and user access, as opposed to network-endogenous factors such as node and link properties which have been extensively studied over the years and handled algorithmically. Policy changes, engendered by the ever-changing business needs, are hard to cope with by algorithms and often dictate manual reconfiguration or reprogramming of WSN, which is difficult because of vendors' complicated and often proprietary implementations. As a result, it is usually inevitable to involve vendors and hence incur delay in policy enforcement, and considerable financial and opportunity cost.

A third problem is that WSN are hard to manage. This is because developing a network management system (NMS) for *distributed* WSN is a demanding task in the first place.

Manuscript received July 31, 2012. The associate editor coordinating the review of this letter and approving it for publication was D. Qiao.

The authors are with the Institute for Infocomm Research, A*STAR, Singapore. T. Q. S. Quek is also with Singapore University of Technology and Design, Singapore (e-mail: {luot, hptan, qsquek}@i2r.a-star.edu.sg, tonyquek@sutd.edu.sg).

Digital Object Identifier 10.1109/LCOMM.2012.12.121712

[1]Throughout this paper, we use "WSN" regardless of its singular or plural form to avoid readers' inconvenience in pronouncing "WSNs".

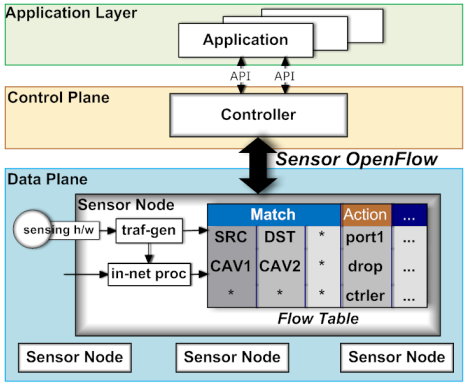


Fig. 1. Software-defined wireless sensor networks.

Furthermore, in practice, this task is often scheduled as "phase-II" in project planning, and hence entails "hacking" *existing* code on sensor nodes, which is a big headache to developers and is highly error-prone.

The above problems are not superficial symptoms but are inherent to WSN and deep-rooted in the architecture: each node is *fully fledged* with all physical up to application layer functionalities, collectively behaving like an *autonomous system* that performs various networking functions such as data forwarding and network control. Although this architecture works well most of the time due to many well-designed algorithms, it lacks good *abstraction* and carries too much complexity, making WSN unwieldy, inelastic to change and hard to manage.

II. SD-WSN and Sensor OpenFlow

In this paper, we take the first move to tackle the above-mentioned problems, using a radical, yet backward and peer compatible, approach. We propose Software-Defined WSN (SD-WSN), an architecture featuring a clear separation between a data and a control plane, and Sensor OpenFlow (SOF), the core component of SD-WSN as a standard communication protocol between the two planes. The data plane consists of sensors performing *flow-based* packet forwarding, and the control plane consists of one (or possibly more) controller that centralizes all the network intelligence, performing network control such as routing and QoS control. This architecture is depicted in Fig. 1. The whole idea is to make the underlying network (i.e., data plane) *programmable* by manipulating a user-customizable *flow table* on each sensor via SOF.

Our proposal is underpinned by the recently emerged software-defined networking (SDN) paradigm [2] and OpenFlow [3], proposed for enterprise and carrier networks. They have become a resounding success, gaining wide support from a large number of industries including Microsoft, Google,

Facebook, HP, Deutsche Telekom, Verizon, Cisco, IBM, and Samsung.

In the academia, related research has recently heated up on a variety of topics [4]–[7]. These studies were all conducted in the context of enterprise and carrier networks alike, such as data centers, where SDN/OpenFlow originated from. In stark contrast, WSN represent a sheerly distinct environment and thereby confront concepts like SD-WSN and SOF with significant challenges.

However, we deem it worth exploring to introduce the fundamental idea of SDN into WSN and tailor it into a viable approach to solve WSN-inherent problems. Indeed, we envisage SD-WSN to transform traditional WSN into networks that are:

• **Versatile**: supporting *multiple* applications in a plug-and-play manner; sensors are no longer application-dependent but application-customizable. This is achieved by (i) the programmable data plane which supports virtually all sorts of packet forwarding rules, and (ii) the control plane which decouples upper applications from physical devices (i.e., sensors) and provides a global, unified view of the underlying network to the applications.

• **Flexible**: easy to enforce policy changes throughout the entire network, which used to be a tedious task and is prone to inconsistency. This is achieved by the centralized and extremely granular network control in SD-WSN, where the granularity is supplied by the fully user-customizable flow tables.

• **Easy to manage**: building an NMS is no different from adding another application on top of the control plane, using open APIs and without hassle of hacking existing code. In addition, the centralized network view provided by the control plane is a great facility, as network administrators prefer centralized NMS in practice.

III. Technical Challenges

The fundamental assumption that OpenFlow makes is that the underlying network is composed of high-speed switches such as Ethernet/MPLS switches and IP routers. OpenFlow was also designed as a *wired* protocol, and its typical use cases are data centers, cellular backhaul, enterprise WiFi backbone, and WANs. Compared to WSN, these represent significant disparities and lead to major challenges in designing SOF.

A. Data Plane: Creating Flows

At the data plane, packets are handled as per flows. A flow is a programmable (i.e., user-customizable) set of packets that share certain properties specified by a *Match* in a flow-table entry, or *flow entry* for short. The Match is often wildcarded, like "IP source address is 10.0.*.*", and packets that match it will be treated as in the same flow and be imposed an *Action* (e.g., "send to port 1") specified by the same flow entry (cf. Fig. 1 or 2).

OpenFlow implicitly assumes the presence of IP-like addressing so as to create flows. For example, it defines Match fields such as `IPv4_SRC`, `ETH_DST` and `MPLS_LABEL` to match packets with IPv4 source address, Ethernet destination address and MPLS label, respectively. However, as opposed to *address-centric* OpenFlow networks, WSN are typically *data-centric*—acquiring data of interest is more important than knowing *who* sent the data—and employ different addressing such as *attribute-based naming*, e.g., "nodes whose sensing temperature>30°C". SOF must cope with this in the first place for flow creation.

B. Control Plane: SOF Channel

An OpenFlow channel is an end-to-end connection used to transmit control messages between a controller and a switch. An SOF channel is similarly defined. However, this channel must provide TCP/IP connectivity [8] for reliable end-to-end in-order message delivery, and the two end-parties are identified using IP addresses. These are generally unavailable in WSN and need to be addressed.

C. Overhead of Control Traffic

SDN can (and usually would) host the OpenFlow channel *out of band*, i.e., using a separate dedicated network. This is normally not practical for WSN and the SOF channel has to be hosted *in band*, i.e., by the WSN itself. Thus, the resource-constrained WSN will have to additionally carry control traffic between controllers and sensors. This is further exacerbated by the fact that control traffic in WSN tends to be large due to the high network dynamics (node/link failures, energy-aware routing, mobility, etc.). Hence, without a proper mechanism to curb control traffic overhead, the underlying WSN can be overloaded.

D. Traffic Generation

End-users are considered peripheral to SDN and hence out of the scope of OpenFlow. On the contrary, sensor nodes behave like end-users by *generating* data packets, in addition to merely forwarding data as OpenFlow switches do.

E. In-Network Processing

At times, WSN need to process data *in-situ*, e.g., perform data aggregation or decision fusion, in order to reduce data redundancy and conserve network resource such as bandwidth and energy. This is another feature absent in SDN.

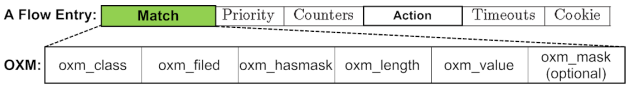
F. Backward and Peer Compatibility

Albeit radical, SOF should desirably provide backward compatibility, with respect to traditional (non-OpenFlow and non-SOF) networks, so as to protect legacy investments.

It is also desirable for SOF to offer *peer* compatibility, i.e., to be compatible with OpenFlow networks which are concurrently being developed and standardized, for interoperability purposes.

IV. Proposed Solutions

In this section, we provide a suite of preliminary solutions to the issues identified above, based on the latest OpenFlow specification (v1.3.0) [8].



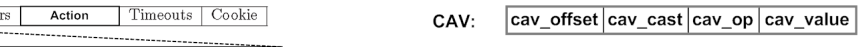
Explanation:

Name		Width	Usage
oxm_type	`oxm_class`	16	Match class: member class or reserved class
	`oxm_field`	7	Match field within the class
	`oxm_hasmask`	1	Set if OXM include a bitmask in payload
	`oxm_length`	8	Length of OXM payload

Fig. 2. A flow entry with `Match` defined in the OXM format.

oxm_type= OXM_SOF_SRC	oxm_hasmask =1	oxm_length =4	oxm_value =0x796F	oxm_mask =0xFF00	oxm_type= OXM_SOF_DST	oxm_hasmask =0	oxm_length =2	oxm_value =0

Fig. 3. Creating Class-1 flows by defining Match using OXM.



Name	Width (bits)	Usage
cav_offset	16	starting position of the attribute (e.g., temperature)
cav_cast	4	data type of the attribute (e.g., int32)
cav_op	4	operator (e.g., >,<,=)
cav_value	determ. by cav_cast	value

Fig. 4. The (new) CAV format with one AV pair; multiple AV pairs shall be concatenated.

oxm_type= OXM_SOF_CAV	cav_offset =48	cav_cast =int32	cav_op =">"	cav_value =30	oxm_type= OXM_SOF_CAV	cav_offset =48	cav_cast =int32	cav_op ="<"	cav_value =60

(a) 30 < temperature < 60

oxm_type= OXM_SOF_CAV	cav_offset =40	cav_cast =int16	cav_op ="="	cav_value =7	oxm_type= OXM_SOF_CAV	cav_offset =42	cav_cast =int16	cav_op =">"	cav_value =150

(b) Zone-ID = 7 and x-coordinate > 150

Fig. 5. Creating Class-2 flows by defining Match using CAV.

A. Data Plane: Creating Flows

We give two solutions for flow creation, from which a network operator can choose based on his own preference. The first solution is to *redefine flow tables* to cater for the special addressing schemes in WSN. We classify WSN addressing schemes into **Class-1**, *compact network-unique addresses* such as the ZigBee 16-bit network addresses (e.g., 0x796F as assigned by the CSkip algorithm [9]), and **Class-2**, *concatenated attribute-value pairs* (CAV) such as "30<temperature<60" and "Zone-ID=7 AND x-coordinate>150".

We handle Class-1 by exploiting the OpenFlow extensible match (OXM), a type-length-value (TLV) format used to define flow Matches, as explained by Fig. 2. In SOF, we introduce two new `oxm_type`'s to represent Class-1 addresses: `OXM_SOF_SRC` (source) and `OXM_SOF_DST` (destination), and construct Class-1 flow Matches under the same OXM framework. A self-explanatory example is given in Fig. 3, where the flow will be composed of packets with source network address of 0x7900–0x79FF and destination address of 0 (ZigBee coordinator).

We handle Class-2 by introducing our CAV format, which is a quadruple defined by Fig. 4. Then, by adding a new `oxm_type`, `OXM_SOF_CAV`, we are able to form any Class-2 flows, as illustrated by Fig. 5, where temperature is assumed to be an int32 stored at offset 48 of each packet, Zone-ID (int16) at offset 40, and x-coordinate (int16) at offset 42.

The second solution is to *augment WSN with IP*, for which we capitalize on and recommend two off-the-shelf IP stacks:

- *uIP* and *uIPv6*: uIP is an IPv4 implementation on Contiki OS [10], an open source operating system for WSN and the Internet of things. uIPv6 is a fully compliant IPv6 extension to uIP and is "IPv6 Ready Phase 1 certified" (has the right to use the IPv6 Ready silver logo). Both uIP and uIPv6 are now part of Contiki OS, contributed by Cisco, Redwire, SAP and SICS.
- *Blip*: the Berkeley IP implementation for low-power networks [11], which is an open source IPv6 implementation for TinyOS based on the 6LoWPAN standard [12]. It has been extensively verified on MicaZ, TelosB, and Epic platforms.

B. Control Plane: SOF Channel

The design of control plane must be consistent with the data plane. Therefore, corresponding to the two solutions above, we also provide two solutions for the SOF channel. If the network operator chooses the non-IP solution, i.e., redefining flow tables, then the SOF channel can be supplied by overlaying a transport protocol directly over WSN. Designing such a protocol has been well studied in the past decade and hence, not to reinvent the wheel, we compile and make publicly accessible a comprehensive list of credible works on WSN transport protocol design, implementation, experimentation and verification [13].

If, otherwise, the network operator chooses to augment WSN with IP (and use our recommended IP stacks), SOF channels will be self-supplied because uIP, uIPv6 and Blip are all shipped with ready-to-use TCP implementations.

C. Curbing Control Traffic

As aforementioned, the control traffic carried by the in-band SOF channel may overload the underlying WSN. Our solution is based on the following observations. First, the control traffic is mainly comprised of two types of control messages: (1) `packet-in` and (2) `packet-out` or `flow-mod`. A `packet-in` is a flow setup request, sent by a sensor to a controller to seek instruction on how to handle an incoming packet upon *table-miss* (a packet does not match any flow entry). A `packet-out` or `flow-mod` is the response from the controller giving instruction to the requesting sensor.

Second, the control traffic resulting from this can be significant, because WSN traffic is often bursty—an event of interest can trigger a large swarm of packets from several sensors, leading to many flow setup requests simultaneously. This scenario repetitively appears because each flow entry is associated with an *expiring timer*.

In our solution, a sensor only sends one `packet-in` for the first table-miss and suppresses subsequent `packet-in` whose associated packets have the same destination address (Class-1 or Class-2) as the first packet, until the corresponding `packet-out` or `flow-mod` is received or a predefined timeout occurs. The rationale is twofold. Firstly, because the end-to-end SOF channel is slow (as it is in-band of WSN), the latency between sending `packet-in` and receiving `packet-out` or `flow-mod` is much larger than in OpenFlow. Therefore, there can be a large number of incoming packets during this interval. Secondly, since the major traffic in WSN is upstream (from

oxm_type= OXM_OF_ETH_TYPE	HM =0	oxm_length =2	oxm_value =0x0800	oxm_type= OXM_OF_IPv4_SRC	HM =1	oxm_length =8	oxm_value =10.0.1.1	oxm_mask =0xFFFF0000

(a) Prerequisite of OpenFlow: a TLV of IPv4 must be preceded by a TLV of `ETH_TYPE=0x0800`.

oxm_type= OXM_OF_ETH_TYPE	HM =0	oxm_length =2	oxm_value =0x0802	OXM_SOF_SRC	1	4	0x796F	0xFF00	OXM_SOF_DST	0	2	0

oxm_type= OXM_OF_ETH_TYPE	HM =0	oxm_length =2	oxm_value =0x0808	OXM_SOF_CAV	48	int32	">"	30	OXM_SOF_CAV	48	int32	"<"	60

(b) OpenFlow-compatible SOF flow entries: a Class-1 entry (upper) and a Class-2 entry (lower).

Fig. 6. Making SOF peer compatible with OpenFlow.

many sensors to one or a few sinks), it is effective to use destination addresses to bundle packets into flows.

In fact, this method can also be applied to (general) OpenFlow networks. We have developed such an algorithm called Control Message Quenching (CMQ) [14] for OpenFlow networks and verified its effectiveness in reducing flow setup latency and elevating network throughput.

D. Traffic Generation

We add a `traf-gen` module on each sensor as shown in Fig. 1. It is a simple module that can be implemented as an interrupt routine, because sensory data generation consists of two very simple steps: reading data from the sensing hardware, and converting the data if needed. An example is given below, based on the Arduino sensor platform [15]:

```
int raw = analogRead(pin); // read raw data from h/w pin
// next, convert to user-friendly data:
// 1) in the case of ultrasound sensors, raw is the measured distance in cm:
float distanceInMeter = raw / 100;
// 2) in the case of temperature sensors, raw is degree Celsius scaled by 20:
float temperatureInCelsius = raw / 20;
// 3) in the case of potentiometers, raw is voltage scaled from 5(V) to 1023:
float voltage = raw * 5.0 / 1023.0;
```

Depending on implementation, this `traf-gen` module can run in a *blocking* (synchronously awaiting sensory data to become available), *callback* (asynchronously triggered by a "data-available" event), or *round-robin* (periodically checking if data are available) manner.

E. In-Network Processing

An `in-net proc` module is added as shown in Fig. 1. If the processing is not needed by a packet, this module simply passes the packet intact to the flow table. In case of possible future changes to the processing algorithm, we use over-the-air programming (OTA), a commercially available technology that allows updating sensor firmware and software wirelessly and remotely. Libelium [16], for instance, offers a package that features secured and interference-free OTA programming over a large network in seconds or up to a few minutes.

We note that in-network processing makes the design not purely "clean-slate", which we deem as a compromise that WSN has to make due to its unique characteristics. In fact, data aggregation is the most common in-network processing and can usually be accomplished using standard operations such as average, median, min, max, or removing redundant data. Therefore, it could be absorbed into flow tables as a special way of packet handling, which would mitigate the compromise and enhance network programmability. We leave this as an open question to the research community.

F. Backward and Peer Compatibility

SOF inherits backward compatibility from OpenFlow. An *SOF-hybrid* sensor will have a `NORMAL` logical port defined similarly as by OpenFlow, which directs packets to traditional sensor network forwarding.

The novelty lies in handling peer compatibility. To be compatible with OpenFlow, SOF should enable OpenFlow to recognize SOF flow tables so that OpenFlow can relegate Class-1 and Class-2 flow entries to SOF. We exploit a prerequisite structure of OpenFlow, which specifies that a protocol-specific OXM TLV, e.g., IPv4, must be preceded by a particular value of `ETH_TYPE` for differentiation purposes. For example, an IPv4 TLV must be preceded by a `ETH_TYPE` of `0x0800` (Fig. 6a), and similarly, IPv6 by `0x86dd` and ARP by `0x0806`.

For the newly introduced Class-1 and Class-2 flows, we designate two unused values of `ETH_TYPE`, `0x0802` and `0x0808`, respectively. Thus, an SOF flow entry (`oxm_type` = `OXM_SOF_SRC/DST` or `OXM_SOF_CAV`) can be defined as exemplified by Fig. 6b, which achieves *peer compatibility*.

V. Conclusion

This paper presents the first effort that synergizes SDN and WSN to solve WSN-inherent problems. The proposed solution, SD-WSN with SOF, might provoke interesting discussions from the research community and open the door to a wide range of innovation opportunities. By that and ultimately, we expect to see a new generation of WSN that are versatile, flexible, and easy to manage.

References

[1] W. Heinzelman, "Application-specific protocol architectures for wireless networks," Ph.D. dissertation, MIT, 2000.

[2] Open Networking Foundation, "Software-defined networking: the new norm for networks," white paper, Apr. 2012.

[3] N. McKeown, T. Anderson, H. Balakrishnan, G. Parulkar, L. Peterson, J. Rexford, S. Shenker, and J. Turner, "OpenFlow: enabling innovation in campus networks," *SIGCOMM Comput. Commun. Rev.*, vol. 38, no. 2, pp. 69–74, 2008.

[4] A. R. Curtis, W. Kim, and P. Yalagandula, "Mahout: low-overhead datacenter traffic management using end-host-based elephant detection," in *2011 IEEE INFOCOM*.

[5] M. Yu, J. Rexford, M. J. Freedman, and J. Wang, "Scalable flow-based networking with DIFANE," in *2010 ACM SIGCOMM*.

[6] P. Dely, A. Kassler, and N. Bayer, "OpenFlow for wireless mesh networks," in *2011 IEEE WiMAN*.

[7] K. Jeong, J. Kim, and Y.-T. Kim, "QoS-aware network operating system for software defined networking with generalized OpenFlows," in *2012 IEEE/IFIP NOMS*.

[8] Open Networking Foundation, "OpenFlow switch specification," Apr. 16, 2012, version 1.3.0. Available: https://www.opennetworking.org/images/stories/downloads/specification/openflow-spec-v1.3.0.pdf

[9] A. Elahi and A. Gschwender, *ZigBee Wireless Sensor and Control Network*. Prentice Hall, 2009, chapter 9.

[10] The Contiki OS, http://www.contiki-os.org.

[11] Blip: Berkeley IP, http://smote.cs.berkeley.edu:8000/tracenv/wiki/blip.

[12] G. Montenegro, N. Kushalnagar, J. Hui, and D. Culler, "IETF RFC4944 (6LoWPAN): Transmission of IPv6 packets over IEEE 802.15.4 networks," 2007. Available: http://www.ietf.org/rfc/rfc4944.txt

[13] T. Luo, H.-P. Tan, and T. Q. S. Quek, "A compiled list of transport protocols for wireless sensor networks," May 2012. Available: http://www1.i2r.a-star.edu.sg/~luot/wsn-tp.pdf

[14] T. Luo, H.-P. Tan, P. C. Quan, Y. W. Law, and J. Jin, "Enhancing responsiveness and scalability for OpenFlow networks via control-message quenching," in *2012 International Conference on ICT Convergence*.

[15] Arduino platform, http://www.arduino.cc.

[16] D. Gascon, A. Bielsa, F. Genicio, and M. Yarza, "Over the air programming with 802.15.4 and zigbee - OTA," May 9, 2011. Available: http://www.libelium.com/over_the_air_programming_OTA_802.15.4_ZigBee/