Title
SOL: An End-to-End Solution for Real-World Remote Monitoring Systems

Authors
Brun-Laguna, Keoma
Watteyne, Thomas
Malek, Sami
et al.

Publication Date
2016

DOI
10.1109/pimrc.2016.7794864

Peer reviewed

SOL: An end-to-end solution for real-world remote monitoring systems

7
Author(s)
Keoma Brun-Laguna ; Thomas Watteyne ; Sami Malek ; Ziran Zhang ; Carlos Oroza ; Steven D. Glaser ; Branko Kerkez
View All Authors

Abstract:
This paper introduces SOL, which is both an efficient data representation for sensor measurements and network statistics, and a complete low-power wireless sensor management system that builds around it. A SOL system consists of multiple low-power wireless mesh networks in which motes connected to sensors and actuators send data to a

single server. It offers multi-tier data replication and the associated data recovery. SOL is used in 3 pilot deployments for micro-climate monitoring, building automation and agriculture applications. In conjunction with Metronome Systems' NeoMote and Manager devices, SOL is a turn-key ready-to-deploy end-to-end system.
Published in: 2016 IEEE 27th Annual International Symposium on Personal, Indoor, and Mobile Radio Communications (PIMRC)
Date of Conference: 4-8 Sept. 2016
Date Added to IEEE *Xplore*: 22 December 2016
ISBN Information:
Electronic ISSN: 2166-9589
INSPEC Accession Number: 16555796
DOI: 10.1109/PIMRC.2016.7794864
Publisher: IEEE
Conference Location: Valencia, Spain

SECTION I.

Introduction

Between 2012 and 2015, the State of California has suffered its most intense drought since record keeping began in 1895. Understanding the detailed effect of the drought on the upper reaches of the Sierra Nevada, the mountain range that spans the eastern border of the state, is important since approximately 2/3 of California's water comes from snowmelt processes. Water plays a crucial role for personal consumption, irrigation of farmland, and production of electricity in hydro-power plants, among others, and it is critical to be able to monitor, explain and predict drought events.

Since 2011, we have deployed and operated 14 low-power wireless networks, to monitor the snowpack in the American River basin [1]. Each network is composed of 10-11 sensor stations deployed over 1 km, ^{2}. A sensor station is a 4 m high pole on which we have mounted snow depth, temperature, solar radiation and relative humidity sensors. Soil moisture and matric suction sensors are inserted into the soil at two depths beneath the station pole. These sensors are connected, through wires, to a low-power wireless module, itself installed in an electrical box on the pole. The wireless modules communicate with one another through a low-power wireless mesh network, and send sensor measurements to a network manager. This manager is connected to the Internet through a cellular or satellite link, allowing it to forward the data to a database running on a server on the UC Berkeley campus. A total of 945 sensors are deployed, each reporting a measurement every 15 min. The data appears in the database near instantly after it was measured.

We use a Metronome Systems' NeoMote[1] as the heart of each sensor station. The NeoMote features a Cypress PSOC microcontroller with 48 fully configurable IO pins, allowing it to interface to virtually all analog and digital sensors. The NeoMote also features a SmartMesh IP low-power wireless module from Dust Networks/Linear Technology[2]. This module handles all the networking out-of-the-box, automatically forming a secure, reliable (over 99.999% end-to-end

reliability) and low-power (<50 μA current draw at 3.6 V) mesh network among sensor stations.

Fig. 1:
SOL has been running for months in three outdoor and indoor pilot deployments.

View All

The network manager sits at the root of the low-power mesh network. We use Metronome Systems' Manager. The Manager is a ruggedized box containing a Linux computer and a SmartMesh IP Manager. It is connected to an embedded satellite modem over an Ethernet wire. The computer runs an application that collects the sensor measurements reported by the sensor stations, and the network statistics generated by the SmartMesh IP low-power wireless mesh network. Every 5 min each mote in the network reports statistics on its own state (charge consumed, battery state, etc.) and on the wireless connections it has to other motes. Collecting and analyzing these statistics is needed to track the state of the network.

One important limitation of the installations is that the satellite connection connecting each manager to the Internet allows only 5 MB of data to be transferred per month. This data includes both the sensor measurements and the network statistics exchanged between the manager and the server.

The “Echo Peak” deployment is representative of the other 13 American River Hydrologic Observatory (ARHO) sites. The Echo Peak network consists of 71 sensors connected to 53 network devices. With a data polled every 15 min; the 71 sensors produce approximately 1 MB of data each month. The network devices generate network statistics each 5 min, which accounts for a total of 10 MB per month.

This highlights how constrained the uplink bandwidth is. The solution that forwards the data from the manager to the server over the satellite link must make a selection of the most representative statistics and cannot induce any overhead itself.

Rather than presenting an extensive comparison with existing solutions, this paper describes the mechanisms we chose to answer the specific application's challenges we faced. This paper introduces and describes the SOL solution, and makes the following contributions:

- We introduce SOL, a lightweight, parsimonious and agile object-based data representation for sensor measurements and network statistics.
- We introduce a complete back-end solution for managing a network of sensors and actuators for real-world monitoring system.
- We contribute a production-ready open-source implementation of the SOL solution to the community.
- We present the results of three pilot deployments using SOL (i) on the UC Berkeley Botanical Garden, for microclimate monitoring, (ii) in the Inria-Paris offices, for a Smart Building application, (iii) in peach orchards in Mendoza, Argentina, to predict frost events.

The remainder of this paper is organized as follows. Section II presents the Sensor Object Library (SOL), an efficient representation of sensor measurements and network statistics, compatible with current standardized representations. Section III describes the end-to-end sensor network management system that uses SOL at its core, and provides sensor data gathering and storage, data retrieval, and remote configuration/reprogramming. Section IV describes three SOL pilot deployments running today. Section V presents the related data representations and protocols, and how they are compatible/-complementary with SOL.
Finally, Section VI concludes this paper and discusses further improvements.

SECTION II.

Sol: Sensor Object Library

The “Sensor Object Library” (SOL) is an efficient representation of sensor measurements and network statistics. It includes loss-less translation between a binary and a JSON format. The binary format is used when sending the

measurements and statistics over the network, and represents the information in a reduced number of bytes. The JSON format is a parsed version of the same data, and is used to insert the information into the database on the server so it can be efficiently queried.

Section V presents an overview of-often standards-based-data representations and associated protocols for constrained (sensor) networks. It indicates how SOL is compatible with other data representations such as CBOR, complementary to IETF CoAP, and how it can integrate with OMA LWM2M or MQTT.

Table I: Fields contained in a SOL object

M	The MAC address of the device which has created the object
T	The timestamp of when the object was created
t	The type of the object, as defined in the SOL registry
L	The length of the value field
V	The value of the object (sensor measurement or network statistic)

Conceptually, a SOL object is a generalized Type-Length-Value (TLV), a format widely used in network protocols. On top of the type and the value of the object, a SOL object contains a unique identifier of the hardware device that has created the object (its MAC address) and a timestamp of when it was created. Table I summarizes the conceptual fields present in each SOL object.

We maintain a registry of the different SOL object types[3]. This registry associates a SOL type (a number) to a format of the value. For example, a SOL object of type 3 (type SNOW_MAXBOTIX_MB7554_RS232_RAW in the SOL registry) represents a raw measurement of the MAXBOTIX MB7554 ultrasonic range-finder, formatted as a 16-bit unsigned integer.

As it implements the IEEE802.15.4 TSCH standard [2], a SmartMesh IP network is synchronized to within 15 µs across the entire network. This allows a sensor station to accurately timestamp when a sensor measurement was taken, before sending it to the manager. The timestamp is included in the SOL object, so even if the satellite link is down, the sensor measurements is still associated with the actual time it was measured when the object is eventually inserted into the database.
The technical format specification of SOL objects is given in [3]. This paper does not repeat that information, but rather describes how SOL objects are used in an end-to-end sensor network management system (Section III), and how this system is being tested on pilot deployments (Section IV). Fig. 2 gives an example of both the JSON and binary formats. We assume mote with MAC address 00-17-0d-00-00-18-ac-50 and mote with MAC address 00-17-0d-00-00-18-22-60 sample a temperature of 26 C and 21 C at timestamps 115364509 and 163263811, respectively. They each report this information to the manager, using SOL objects with type 33 (temperature). The manager reports this to the server, as a compound SOL object.

Fig 2a shows the binary representation of the compound SOL objects. The binary SOL representation allows the compaction of compound SOL objects: fields that have the same value in individual objects (the SOL type in the example) only appear once in the SOL compound object to save space. The binary representation is used when objects transit from the motes to the server.

Fig 2b shows the JSON representation of the compound SOL object from Fig 2a. Both JSON and binary formats are equivalent and translation from one format to the other is lossless. Upon receiving the data in binary format, the server translates it into the equivalent JSON format, which it inserts into the database. This fully parsed format allows queries to be done on the database for post-processing of the data.

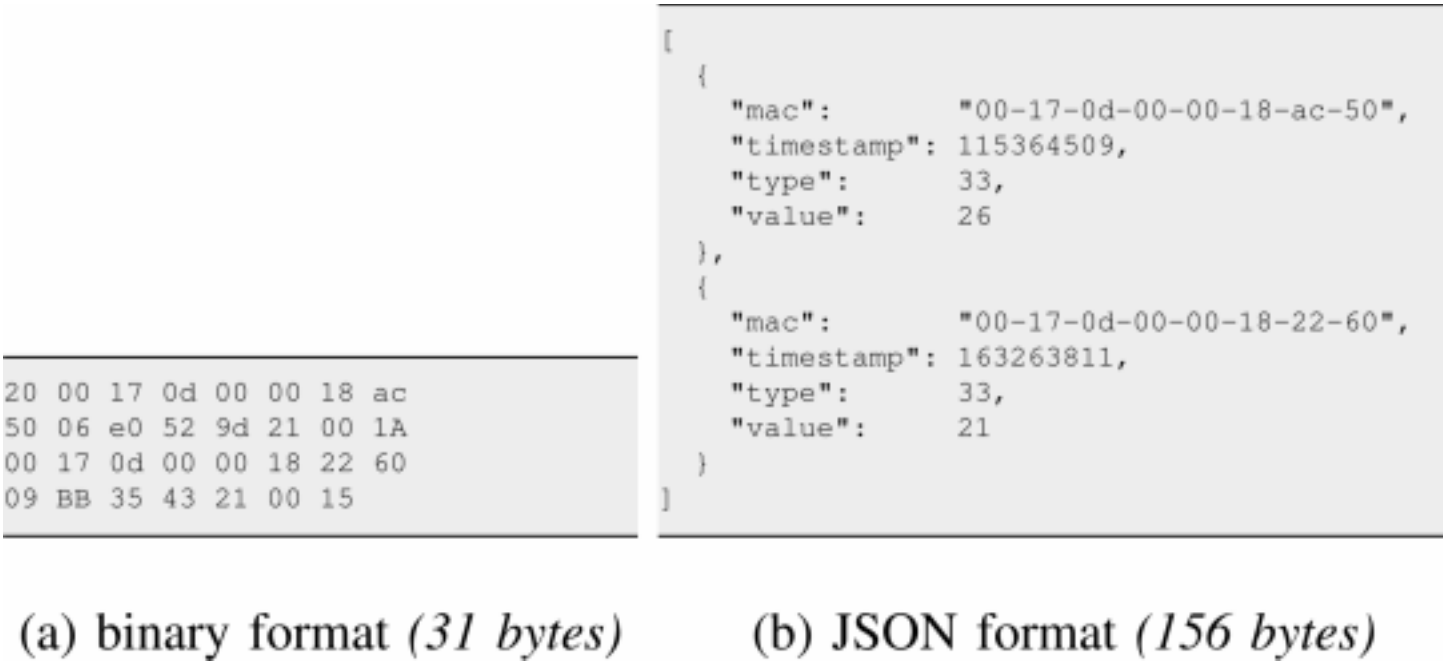

Fig. 2:
The binary and JSON formats of an example compound SOL object.

View All

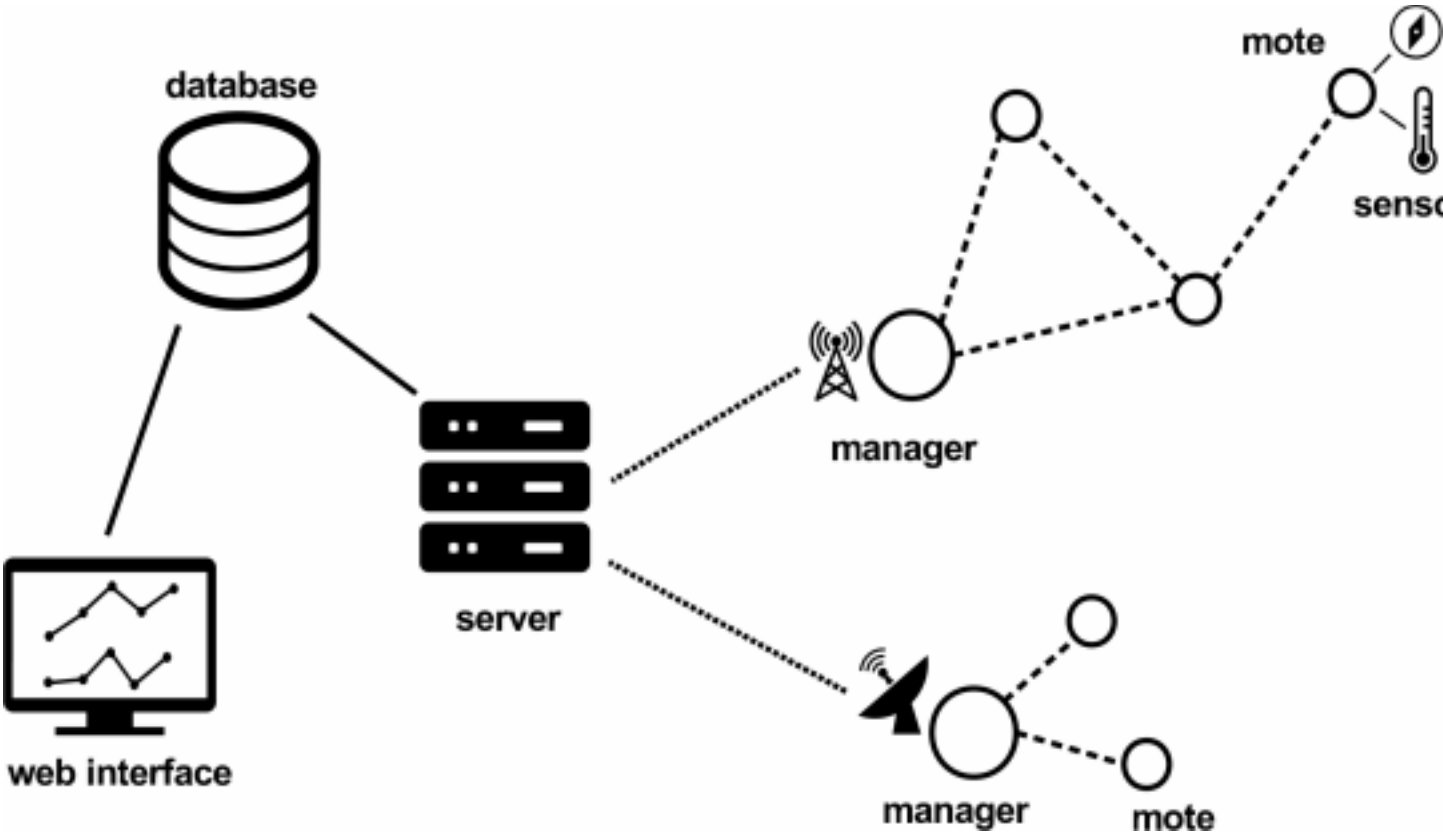

Fig. 3:
End-to-end network architecture of a SOL system.

View All

SECTION III.

End-To-End Integration and Implementation

Fig. 3 depicts the overall end-to-end architecture of SOL. Each mote samples each of the sensors it is attached to every 15 min and creates the corresponding SOL objects in binary format. The SOL objects are reported to the manager through the SmartMesh IP low-power wireless mesh network. If multiple sensor measurements are reported at the same time, the compound representation is used. The manager further reports the SOL binary objects to the server over a satellite link. The server translates the SOL binary objects into their JSON equivalent format, and enters that information into the database. The server presents that data to a user through a series of dynamic web pages.

A. Implementation

The implementation of the end-to-end SOL sensor network management system consists of three elements: mote firmware, manager firmware and server software.

1) Mote Firmware

These elements are implemented as PSOC Creator modules. PSOC Creator is the Integrated Development Environment provided by Cypress, the manufacturer of the PSOC micro-controller on the NeoMote. PSOC Creator offers a graphical interface allowing the user to drag-and-drop modules and configure the use of the IO pins. The SOL-specific code, which performs the tasks listed below, is implemented as a module and can hence be dragged-and-dropped into a user's project. This gives users entire flexibility, and in particular allows them to integrate any sensor or actuator through the PSOC's advanced interfacing options. The SmartMesh IP module on the NeoMote periodically generates network statistics without requiring intervention from the PSOC micro-controller. The firmware on both the SmartMesh IP module and PSOC micro-controller can be remotely replaced using Over-the-Air-Programming (OTAP) capabilities bundle in both the SmartMesh IP and PSOC products.

The mote firmware runs on the PSOC micro-controller of the NeoMote. It is written in C and is responsible for (i) sampling the sensors attached to the mote, (ii) formatting those measurements as SOL objects, (iii) making a local copy of that information, (iv) interfacing to the SmartMesh IP mote over a serial (HDLC) interface to send the data to the manager, and (v) offering an interface for remote configuration, for example to change the rate at which a sensor is sampled.

2) Manager Firmware

The manager firmware runs on the GNU/Linux computer of Metronome Systems' Manager. The firmware is written in Python and is responsible for (i) interfacing to the SmartMesh IP manager over a serial (HDLC) interface, (ii) retrieving the sensors measurements and network statistics generated by the low-power wireless network, (iii) making a local copy of that information, (iv) forwarding that information to the server, and (v) offering an interface for the server to configure the manager, and retrieve sensor measurements and network statistics over some range of time.

The manager firmware is implemented as a multi-threading Python application. It connects to the server over a RESTful interface, implemented as JSON over HTTPS. Authentication between the server and the manager is made using certificates. Authorization is done using tokens.

3) Server Software

The server software runs on a GNU/Linux server, hosted by UC Berkeley for the pilot deployments listed in Section IV. It consists of (*i*) a Python application that offers a RESTful *HTTPS/JSON* interface for the manager to publish data, (ii) an InfluxDB database to hold the SOL objects parsed as time series, (iii) a (mostly Grafana-based) web visualization framework for users to visualize the data in real-time.

The source code of the different components of a SOL system are provided open-source under a BSD license [4]. State-of-the-art source control, continuous

integration, deployment and bug tracking are used to maintain production-quality source code, ready to be deployed.

B. Deploying Sol

1) Create Sensor Stations

The first step of deploying SOL is to create sensor stations, by connecting the NeoMotes to sensors. This step is the most open-ended, as each application requires different sensors and different measurement settings. A user must select the sensors and connect those to the NeoMote.

The hardware integration is greatly simplified by the programmable pinout of the PSOC, which allows pins on the micro-controller to be re-purposed as analog and digital interfaces. Besides re-purposing pins, the PSOC comes with a number of analog and digital blocks internal to the chip, which can be “wired” to interface to virtually any sensor or actuator. Analog blocks include switch capacitors, opamps, comparators, Analog-to-Digital converters, and Digital-to-Analog converters. Digital blocks include timers, counters, Pulse Width Modulation (PWM), or serial communication blocks.

The software integration is greatly simplified as the SOL-specific mote firmware is provided as a PSOC Creator module. As detailed in Section III-A1, this module formats SOL objects and interfaces to the SmartMesh IP mote. It can be added to a customer's application in a drag-and-drop fashion.

Note that the low-power wireless mesh network can consist of any SmartMesh IP devices, including non-NeoMote boards.

2) Create the Manager

The manager firmware runs on any GNU/Linux platform that runs a Python interpreter. The preferred hardware is the Metronome Systems' Manager, as is already integrates a SmartMesh IP manager, and has been designed to run in harsh environments. Creating the manager consists in running the latest release of the manager firmware, configuring the address of the server, and provision the server certificate and authorization token. When switched on, the manager attempts to forward all sensors measurements and network statistics to the server, by encoding both as a JSON strings. The certificate ensures the server authentication and allows this JSON string to be sent securely over HTTPS. The token, sent as an HTTP header, allows the server to authorize the manager.

3) Create the Server

The server software runs on any GNU/Linux platform that runs a Python interpreter. Creating a server consists in running the latest release of the software, and ensuring that the manager(s) can issue HTTPS requests to the server.

4) Deploy the Low-Power Wireless Network

Deploying the low-power wireless network consists in switching on the manager and the sensor stations, and letting the network form. The SmartMesh IP firmware on the manager and motes handles network formation and self-healing, thus, no action is required from the user. The devices publish statistics about the connectivity between each others. These statistics are forwarded to the server and allow a user to track the “health” of the network. If the network is too sparse, additional repeater nodes can be installed.

5) Process Incoming Sensor Measurements

This last step is open-ended, as how to process the sensor measurements depends on the application. The server software offers a web-page dashboard to visualize the time series of sensor measurements as it is received. The data can simply be logged, displayed in a convenient manner, fed into prediction models, etc. Section IV details how the sensors measurements are used in three pilot deployments.

C. Miscellaneous Features

1. *Support of multiple mesh networks:* The SOL solution allows multiple low-power mesh networks, each with its own manager, to report to the same server. A secure connection is established between each manager and the server using HTTPS. Adding a new low-power mesh network is straightforward, and consists in providing the server address and certificate to the new manager.

2. *Impact of packet size on energy consumption:* In a SmartMesh IP low-power wireless mesh network, the maximum application payload is 90 B. Because of the underlying technology, it is more energy efficient to send one packet with 90 B of payload, rather than 9 packets with 10 B of payload. The recommended behavior of the sensor station is to collect sensor measurements and to send multiple measurement as one SOL compound object, as close as possible to 90 B. Since each measurement is timestamped, no information is lost even if the sensors measurements are buffered on the sensor station. The implementer can choose to implement a timeout such that sensor measurements are sent to the manager even when they don't occupy 90 B of the payload.
3. *Periodically transmitting from manager to server:* Maintaining a permanent TCP session between the manager and the server is not feasible as the satellite connection of the manager often breaks. Instead of forwarding the data as soon as it receives it, the manager locally buffers the sensor measurements and network statistics, and transfers them as a compound SOL object periodically.

4. *Multi-tier data replication:* Each element in the network can fail, and bringing it back online can take time. To avoid loosing data, data is backed up at different locations. All sensor measurements are stored onto a flash

memory card at each NeoMote before being sent to the manager. Sensor measurements and network statistics are stored to non-volatile memory at the manager. The data can hence be reconstructed in the event of catastrophic failure of different parts of the network.

5. *Retrieving old data:* The server can retrieve old data from the multi-tier data backups in an automated way. The manager offers a RESTful interface that allows the server to retrieve the data stored in the manager back-up between two dates.

6. *Changing environment parameters:* The server can send commands to the manager to change the environment parameters such as calibration constants, sampling intervals, sensor operational status, and programming the operational real-time clock after the SOL system deployment.

SECTION IV.
Pilot Deployments

SOL has been running for months in real-world pilot deployments. The role of those pilots is to increase the level of maturity of the code, and assess how SOL matches the needs of real use-cases.

A. Precision Agriculture, Botanical Garden, UC Berkeley, USA

The UC Berkeley Botanical Garden has over 13, 000 different kinds of plants from around the world, cultivated by region in naturalistic landscapes over its 34 acres. Established in 1890, it is a landmark of the University.

Irrigation in such as diverse botanical environment is complex, in particular as different plants have different needs. The team of botanists combine meteorological information with observations of the plants. The goal of this pilot is to complement that with detailed in-situ measurements. In 2015, we deployed a SOL network with 5 motes and 8 repeaters [56]. The sensor stations are equipped with air temperature, relative humidity, soil moisture, soil temperature and soil electrical conductivity sensors at two depths. The manager is connected to a server hosted at UC Berkeley through a cellular connection.

Both real-time and historic sensor measurement data is made available to the team of botanists in charge of the garden.

B. Building Automation, Inria Offices, Paris, France

The Paris campus of Inria opened in January 2016. With 540 researchers and staff divided over 40 research teams, it is the largest Inria site. The Inria-Paris campus consists of two large 6-story buildings, which feature state of the art building automation. Heating, ventilation, air-conditioning, blind operation and

lighting are operated through IR remote controls deployed throughout the building.

To further automate the operation of the building, IR remote controls (which require manual operation) are being augmented by low-power wireless motes equipped with IR transmitting capabilities. These motes allow tasks to be automated, including shutting the blinds when the illumination gets too high, or starting the heating right before the occupant of an office arrives. An initial proof-of-concept deployment of a SOL system has been rolled out, and consist of 2 managers and 6 motes. The data is sent to a local server and can be visualized in real-time 6. Fig. 4 shows an example dashboard displaying the temperature measurements and network statistics over a 7 days period.

C. Frost Prediction in Peach Orchards, Mendoza, Argentina

In 2013, 85% of the peach production in the Mendoza region (Argentina) was lost due to frost events. The flowering period of the peach trees (September in Argentina) is a critical period. If during that period, temperature drops below-3 C for over 4 hours, the flower buds fall off and no peaches are produced. This means that a farmer can lose most of his/her crop in one night. If the frost event is detected, effective countermeasures exist, such as using return-stack orchard heaters and a helicopter to mix hot and cold air. Detecting those events is typically done by combining the in-situ data from a (single) meteorological station in the orchard with meteorological data. This leads to false positives/negatives, as the micro-climatic variance inside the orchards cannot be captured by a single meteorological station.

Fig. 4:
Example dashboard for visualizing sensor the measurements and network statistics from the inria smart building pilot deployment in real time.

View All

To accurately model the micro-climatic variance in the orchard, we have equipped three trees with temperature, solar radiation, wind velocity and direction and relative humidity sensors in a 110 m×50 m peach orchard in the Mendoza region [7][8][9]. A total of 23 motes have been deployed and report sensor values every 30 seconds. The collected sensor measurements are fed into a micro-climatic model that uses machine learning to predict frost events.

SECTION V.

Related Data Representations and Protocols

Several standards-based sensor data representations and associated protocols exist. This section discusses why SOL was created, and how it is compatible with those existing solutions.

Virtually any binary protocol organizes the bits and bytes of its header into fields. The binary format of a SOL object adopts the same approach, and defines a header (containing the MAC, timestamp, type and length fields) and a payload (the value). Since a SOL object can also be represented as JSON object, it is tempting to use already existing binary representations of a JSON object instead of a custom header format.

A. Data Representation

There are several (often standards-based) binary JSON representation. UBJSON [8] and Smile [9] focus on keeping the binary representation human readable. The “Concise Binary Object Representation” **(CBOR)** [4] is designed to translate JSON to binary format with minimal computation overhead.

The reason why SOL does not use a generic binary representation of JSON is because it would result in longer binary representations. To verify this, we extract 2 weeks worth of network statistics in JSON format from the server on the Inria pilot deployment (Section IV-B), and apply the different binary translations. Results are summarized in Table II. The SOL binary format offers greater compression compared to the generic UBJSON, Smile and CBOR. This is expected, as SOL binary is custom-made to encode only SOL objects; other solutions are generic and can encode any JSON string.

Table II: A two-weeks dataset of JSON SOL objects encoded using different binary formats

Format	JSON	UBJSON	Smile	CBOR	SOL
Size	927 kB	612 kB	497 kB	408 kB	289 kB
Compression	-	34%	46%	56%	69%

It is important for a SOL binary object to be encoded in as few bytes as possible since it saves energy in the low-power wireless mesh network, and bandwidth on the satellite link. We therefore use the SOL binary format between the mote and the manager, and between the manager and the server. This format can easily

be translated to CBOR at the server, for example if the data is made available through a CoAP proxy.

B. Communication Protocols

SmartMesh IP implements 6LoWPAN and is hence IPv6 ready. The motes in the SOL implementation send the SOL object to the manager directly over UDP, to port 0xf0ba. One legitimate question is: “Why not use an application-level protocol to carry the SOL objects”.

The “Constrained Application Protocol” (CoAP) [5] is an ideal candidate. CoAP is a standards-based RESTful protocol designed for constrained devices, often dubbed the “HTTP for motes”. As such, it offers services similar to HTTP, including resource identification through URIs, and timeout/retry. Several extensions to CoAP exist: 1) OMA Lightweight M2M **(LWM2M)** [6] that standardizes the format of the CoAP payload. 2) the IPSO Smart Object Guidelines [7] that define well-known sensor/actuator types based on the OMA LWM2M object model. 3) ETSI M2M that defines the requirements, functional architecture and interface descriptions for M2M communications (as part of the OneM2M initiative [10]). In their paper [8], F. T. Mamo et al. implement such extensions and discuss their strength and weaknesses.

While those solutions offer powerful mechanisms such as data management and resources identification, some features are not mandatory for our application. The satellite connection in a SOL system often drops, so data is not sent directly from the mote to the server. Using an end-to-end protocol such as CoAP hence does not apply to SOL. Moreover, a SmartMesh IP network is reliable; complex retransmission at different levels of the protocol stack ensure that a data packet generated by a mote makes it to the manager. This means the timeout-and-retry service that CoAP offers through Confirmable Messages (i.e messages that require an acknowledgment) is not needed. Finally, data generated by a mote sent to UDP port 0xf0ba are always SOL objects, thus, the concept of CoAP resources is not needed.

What is done with the data at the server depends entirely on the application. The server can make the data available over popular publish/subscribe protocols such as “Message Queuing Telemetry Transport” **(MQTT)** [9] or a CoAP proxy, can inject the sensor data in a SCADA management system, or publish to a third-party cloud service such as Xively. Using SOL for efficiently communicating the data from the mote to the server does not prevent any of these interactions.

SECTION VI.

Conclusion

This paper presents SOL, which is both a lightweight object-based data representation, and an end-to-end low-power wireless sensor network management solution. SOL is fully implemented and available open-source as ready-to-deploy and production-ready code. SOL has been running for months

on micro-climate monitoring, building automation and agriculture pilot deployments. The data representation and protocol used in SOL are designed towards low-overhead, but stay compatible with popular binary representations such as CBOR or protocols such as CoAP.

Using a manager for application-level forwarding between the motes and the server opens up the solution to man-in-the-middle attacks. We are currently working on implementing object-security on top of SOL object, using the solutions designed by the IETF COSE working group, inspired by OSCAR [10].

Our next pilot deployment, scheduled for summer 2016, is to retrofit the 14 Sierra Nevada deployments with SOL.