A Demonstration of the NITOS BikesNet Framework

Giannis Kazdaridis, Donatos Stavropoulos, Stavros Ioannidis,
Thanasis Korakis, Spyros Lalis and Leandros Tassiulas
Department of Electrical and Computer Engineering
University of Thessaly, Greece
Centre for Research and Technology Hellas, CERTH, Greece
Email: {iokazdarid, dostavro, iostiv, korakis, lalis, leandros}@uth.gr

***Abstract*—In this paper we present NITOS BikesNet, a framework for mobile sensing in a city-wide environment offering experimentation capabilities. More Specifically, we present a custom-made and modular prototype device that can be easily mounted on volunteers' bicycles dedicated to collecting environmental measurements and available WiFi networks. In addition, we present our enhancements in OMF framework through which we remotely control the operation of the developed devices, whenever they experience back-end connection. Finally, we analyze an indicative demonstration experiment which illustrates the capabilities of the developed framework.**

I. Introduction

Smart and portable devices such as tablets, smart-phones and other wearable devices have infiltrated in everyday life forming a powerful tool that enables communication and ubiquitous access in the information systems. Those devices feature a high number of sensing modules that allow for capturing real data as well as uploading and sharing them with the interested parties. This type of community sensing is widely known with the term of participatory sensing [1].

In this paper, we present the NITOS BikesNet framework, which expands NITOS facility [2] capabilities, using sensor nodes mounted on bicycles. Our platform is based on a custom wireless embedded node that can host different types of sensors. The sensor nodes are mounted on the bicycles of ordinary people who go about their daily routine as usual. The management of the sensor nodes and the data produced by them is done via the cOntrol and Management Framework (OMF) [3] and the OMF Measurement Library (OML), respectively. Leveraging our previous experience in delay tolerant networking (DTN) [4], [5], we extend OMF to handle intermittent and disconnected operation due to mobility, in a transparent fashion.

A similar work [6] to our implementation uses a variety of sensing modules, which focuses on environmental as well as cyclist's performance/fitness measurements and is based on several Moteiv Tmote Invent platforms. The platform is modular and provides experience mapping, but is not designed for allowing remote access and is not controlled by a widely adopted framework. Moreover, the commercial devices used in this platform do not offer flexibility in terms of customization, whereas in our work we relied on open-source modules, which allow fine-grained configuration. In the domain of Smart Cities experimentation SmartSantander [7] constitutes an indicative example, since it features experimentation on a smart city infrastructure with fixed and mobile nodes. The provided interface and tools are custom made for the purposes of SmartSantander and heavily rely on its architectural components.

This work is organized as follows. Section II presents the overall architecture of the framework and Section III presents an experimental demonstration scenario that showcases NITOS BikesNet capabilities.

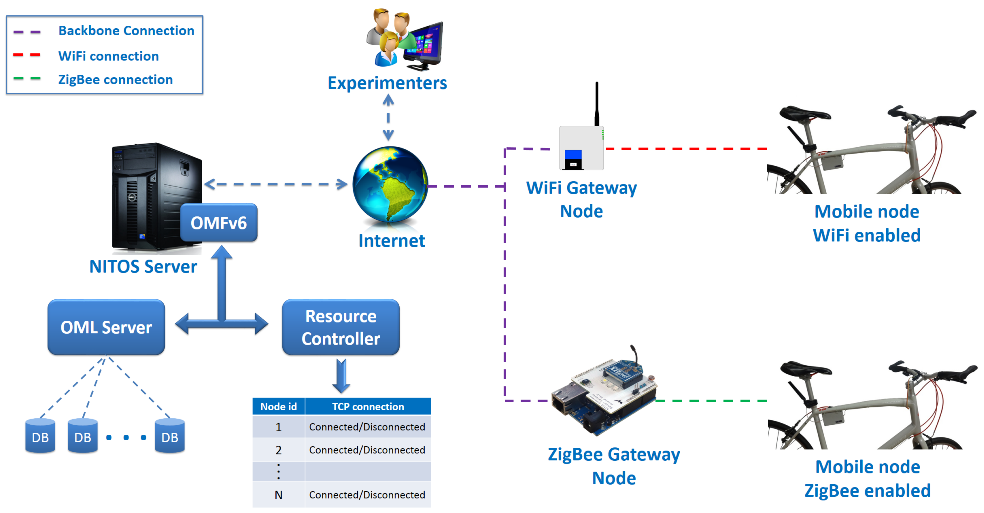
II. NITOS BikesNet Framework

The NITOS BikesNet Framework enables experimentation in a mobile wireless sensing network consisting of sensing devices mounted on bicycles. The deployment of the framework was carried out as part of the NITOS infrastructure and the management of the experimentation is handled by the OMF framework which allows the experimenters to describe their experiments in an event-driven way using a high-level description language.

The basic components of the OMF framework are the Experiment Controller (EC) and the Resource Controllers (RCs). The role of the EC is to orchestrate the execution of the experiments, written in the OMF Experiment Description Language (OEDL). The EC interprets OEDL and sends appropriate messages to the corresponding RCs. In turn, each RC is responsible for abstracting and controlling one or more underlying physical or logical resources. It basically converts the messages received from the EC into resource-specific commands, and relays the response back to the EC. It is important to note that the message exchange between the EC and the RCs is performed using a publish-subscribe mechanism, assuming a stable and reliable communication. Thus, in case of network problems, the messages published by the EC and/or the RC are dropped.

A. Framework Architecture

The NITOS BikesNet platform consists of three distinct physical entities: the server, gateways and mobile sensor nodes. The NITOS server runs an instance of the OMF/OML framework, appropriately extended to handle node mobility (see next). Gateways provide wireless Internet connectivity to the sensor nodes. They are placed/fixed at different areas of the city and have a backbone connection to the public Internet. There are currently two gateway types: WiFi access points / routers, and custom-made NITOS ZigBee access points. WiFi is a widely adopted standard with numerous APs available in every city, while ZigBee has a lower power consumption and thus could be an interesting alternative to explore for nodes that run on batteries; also, ZigBee operates at several different frequencies and can avoid the interference in the 2.4GHz unlicensed band induced by WiFi networks. Finally, NITOS mobile sensor nodes are mounted on the bicycles of volunteer participants. They are implemented using a self-designed embedded device and custom firmware that controls



Node id	TCP connection
1	Connected/Disconnected
2	Connected/Disconnected
⋮	
N	Connected/Disconnected

Fig. 1: NITOS BikesNet architecture.

the onboard sensors, logs measurements on stable storage and manages the communication with the server, via the gateways. Fig. 1 illustrates the system architecture.

To deal with the intermittent connectivity and disconnected operation of mobile sensor nodes, NITOS BikesNet employs a custom-developed RC component, which performs the actual communication with the nodes while hiding any disconnections from the rest of the NITOS platform; as before, the RC itself has a stable connection to the EC and OML components. More specifically, in addition to conventional (synchronous) commands, the RC supports so-called *asynchronous* commands. A conventional command succeeds only if the node to which it is addressed is available (i.e., is currently connected to the server through a gateway), else it fails; this corresponds to the usual operation of the RC. In contrast, an asynchronous command is deferred if the corresponding node is not available; it is queued within the RC, and is forwarded to the node as soon as it becomes available. Asynchronous commands can be associated with an expiration time, after which they are dropped from the queue; they are also removed from the queue if in the meantime the user issues a new, competing command that effectively cancels them. Moreover, when a mobile sensor node becomes available, the RC retrieves its measurements and forwards them to an OML collection point.

B. *Sensor node protocol*

The high-level protocol between a mobile sensor node and the RC is outlined in Table I. The conversation is initiated by the node, which announces itself to the RC via a "HELLO" message. Conversely, it sends a"BYE" message when it intends to end the conversation; there is no negotiation here, as the node unilaterally decides to sign-off. But note that the conversation may also end abruptly, without the node sending a "BYE" message. The transfer of commands to the node and measurements/results to the RC is done via the "CMD" and "RSLT" transaction, respectively. In both cases, the other side must confirm the receipt and successful handling of the message via an acknowledgment ("CMDACK" or "RSLTACK"), else the transaction will be considered as failed (and the message will be eventually retransmitted). All protocol messages carry the node identifier in order for

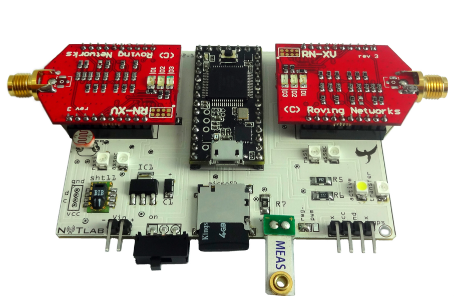
Direction :: Message	**Comments**
N $\rightarrow$ RC :: $<$node id$>$:HELLO	Node announces its availability. The RC should update the node status internally. Initiate transmission of pending commands from the RC to the node, as well as transmission of pending results from the node to the RC.
RC $\rightarrow$ N :: $<$node id$>$:CMD:$<$seq. #$>$:$<$payload$>$ N $\rightarrow$ RC :: $<$node id$>$:CMDACK:$<$seq. #$>$	RC sends the next pending command, and waits for an acknowledgment. Can be repeated several times.
N $\rightarrow$ RC :: $<$node id$>$:RSLT:$<$seq. #$>$:$<$payload$>$ RC $\rightarrow$ N :: $<$node id$>$:RSLTACK:$<$seq. #$>$	Node sends next batch of results to the RC, and waits for acknowledgment. Can be repeated several times.
N $\rightarrow$ RC :: $<$node id$>$:BYE	Node informs that it will sign off and become unavailable. The RC should update the node status internally.

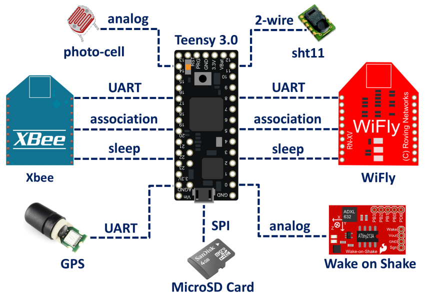
TABLE I: Messaging protocol for the communication between the resource controller (RC) and the node (N).

the RC to associate them with the corresponding internal data structures. In addition, command/result messages and the respective acknowledgments carry sequence numbers for association and duplicate detection purposes.

Our current implementation uses TCP/IP as the underlying transport service, because of its support for reliable communication, flow-control and full-duplex bi-directional data transfers. Mobile nodes are pre-configured with the IP address and TCP port number of the RC (which runs on a publicly accessible machine with a static address). Nodes connect to the RC when they encounter a gateway and have not communicated with the RC for some time. The connection may remain open for a longer period, in which case the node receives the new commands from the RC as soon as these are issued from the user, and sends new measurements to the RC as soon as these are performed. A node will close the connection when all measurements have been successfully transferred to the RC, and it has not received a new command from the RC for some time. Of course, the connection can brake at any point in time during the above transfers, as nodes may shut down, reboot, or go out of range of the gateway.

Note that the protocol does not make any strong assumptions about the underlying transport. Thus it can be applied, virtually unchanged, on top of different transports (besides TCP/IP). In fact, we exploit the "portable" nature of the protocol in how we interface the mobile sensor nodes to the RC via ZigBee (as will be discussed in the sequel).


(a) Mobile sensor node



(b) Component diagram

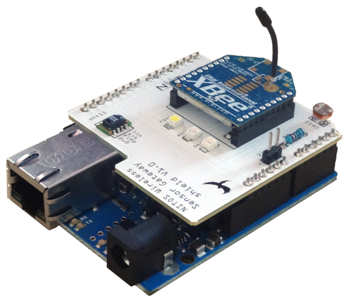

(c) ZigBee gateway

Fig. 2: Custom-built hardware of the NITOS BikesNet platform

C. Developed hardware and firmware

The NITOS BikesNet sensor node, depicted in Fig. 2(a), is built by combining different hardware components to create a unified solution. Our prototype is based on the Arduino platform [8] and several Arduino compatible modules because of their open approach, great flexibility and the large number of publicly available hardware modules and software libraries. A high-level component diagram of the sensor node is shown in Fig. 2(b).

Teensy microcontroller board: The main module of the node is a Teensy 3.0 development board [9], which embeds a 32-bit ARM Cortex-M4 microprocessor. Teensy features 34 digital I/O pins (14 of those can be configured as analog ones) and 3 UART ports. It operates at 3.3V, while the microprocessor can be configured to run at 96MHz, 48MHz, 24MHz or 2MHz. Furthermore, Teensy can be put in sleep mode to minimize power consumption. Notably, Teensy is fully compatible with Arduino software, and it can be programmed using the same tools.

Wireless interfaces: In order to give the node sufficient flexibility in terms of networking, it has two sockets for plugging-in communication modules. These are connected to the Teensy via two different UART ports. Each interface has separate hardware connections that are used to control the power state (put the modules in sleep mode) and to monitor the association status of the respective network interface.

We currently employ two modules that support different popular wireless standards: the Xbee Series 2 module [10], and the WiFly RN-171 module [11]. The Xbee module implements the ZigBee protocol on top of 802.15.4 in the ISM 2.4GHz band. The WiFly module implements the 802.11 b/g radio protocol and is compatible with commercial WiFi access points. In our current implementation, the Xbee is used to associate with ZigBee gateways through which the node can communicate with the RC. Of course, the WiFly also serves this purpose. In addition, it can be used as a sensor, to scan available WiFi networks. Different node configurations are possible. For instance, one may have a WiFly module acting as a network sensor, and an Xbee module or a second WiFly module for the communication with the RC.

Sensors: The node is equipped with a GPS unit and temperature, humidity and light-intensity sensors. The GPS is a D2523T receiver [12] that provides geo-positioning as well as ground speed and altitude. It also provides exact time/date information, which in turn can be used to timestamp measurements in a globally consistent way. The GPS is interfaced with Teensy through a UART port.

The air temperature and humidity is measured via a sht11 digital sensor [13]. Finally, a photocell sensor that consists of a light-dependent resistor is interfaced with one of Teensy's analog I/O pins, providing luminosity measurements.

microSD: The node also features a microSD slot connected to Teensy. The microSD is used to persistently log sensor measurements, until they are uploaded to the RC. This way the node is able to collect a large amount of data without having network connectivity to the RC.

Wake-on-shake: The Wake-on-Shake (WoS) [14] combines a low-power microcontroller with the ADXL362 accelerometer and has extremely low power consumption. It provides an analog pin that is activated in case of an abrupt shake (strong acceleration). We use WoS to awake Teensy as soon as the mobile node starts moving, in order to initiate the sensor measurement process. Note that the Teensy cannot acquire vibration measurements by itself when in sleep mode.

Firmware: The firmware of the NITOS BikesNet sensor node controls the entire operation of the device, e.g., it configures and controls the peripheral modules, performs sensing tasks, logs the respective measurements, implements power management, etc.

In a nutshell, as long as the node is moving, the Teensy periodically executes a full measurement cycle. If the node stays immobile for a longer period of time, Teensy puts itself and the wireless interfaces in sleep mode while it completely turns off the rest peripherals apart from the WoS. Normal operation is resumed when Teensy is woken up by WoS. To use the WiFly as a sensor (for detecting the WiFi networks in range), it is put in command mode and is instructed to perform a scan operation. When the scan is completed, the WiFly responds with the list of available networks. In case of using a network module (WiFly or ZigBee) as a communication channel, the firmware monitors the respective interface to detect that a connection to a gateway has been established. Then, it initiates a conversation with the RC (as described previously). As said, the Xbee module is used exclusively for the purpose of communicating with the RC (via a NITOS ZigBee gateway). But unlike WiFly, Xbee does not come with built-in support for TCP/IP. Fortunately, the firmware can talk the same high-level protocol on top of the Xbee application-level transport service; in turn, the ZigBee gateway acts as a proxy for the node over a TCP connection to the RC. The association with NITOS ZigBee gateways is done using a pre-configured PAN id.

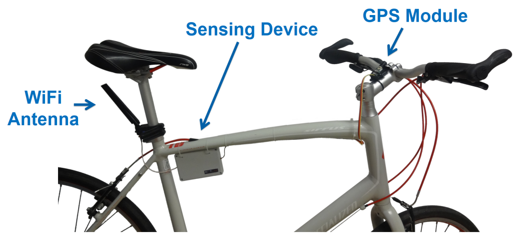


Fig. 3: NITOS mobile sensor node mounted on bicycle

ZigBee gateway: As already mentioned, we want a mobile node to be able to communicate with the RC not only via WiFi but also via ZigBee. The respective gateway, illustrated in Fig. 2(c), is custom built for this purpose. It is based on an Arduino Ethernet board, on top of which we mount a custom shield with an Xbee module. The Ethernet is connected to a backbone network through which the RC can be reached. The Xbee is configured as a coordinator for a given PAN id. When a node associates with the ZigBee network of the gateway, the gateway initiates a TCP connection to the RC on behalf of the node. From that point onwards, the gateway receives (over the Xbee transport) the protocol messages sent by the node, and forwards them (over TCP/IP) to the RC. The same is done for the messages flowing in the other direction.

Installation of the NITOS sensor node: The sensor node, enclosed in a waterproof case, is attached to the bicycle using tire-ups. The antenna for the WiFi radio is fixed beneath the saddle, and is connected via a pigtail to the WiFly interface of the node. Finally, the GPS unit is placed on the handlebar of the bicycle in order to have good signal reception. Fig. 3 illustrates the complete setup on one of our bicycles.

III. Experimental Demonstration

In this section, we analyze a representative experiment that demonstrates the innovative potentiality of the developed framework. For the purposes of this experiment, a couple of sensing devices will be used for local demonstration of the framework's hardware. Due to constraints the GPS module will not be used as the demonstration will be static and in an indoor environment. However, the rest of the modules will be fully operational and the movement of the bicycle will be emulated manually as the same will be done for the network disconnections that usually occur in a realistic experiment.

Initially, we describe the experiment scenario in an OMF script, in which we clearly denote the timings of the experiment as well as the modules we want to enable and obtain measurements from. An example of a command can be seen in the Listing 1 where the OMF will send the command 'StartDevice' to the device with id number '2' after 6 seconds of the experiment starting time. During the experiment execution, OMF will relay commands to the sensing device so that its modules will start measuring the surroundings. Several features of the framework will be demonstrated, like the Wake-on-Shake (WoS) module which helps in saving energy by making the device to "sleep" when there is no movement of the bicycle for a specific duration of time. Regarding the abrupt disconnections, those will be emulated manually by disconnecting the wireless interface and reconnecting it back. These actions correspond to the movement of a bicycle in a city, where network connectivity is in general fragmentary and the movement of a bike is disrupted due to traffic conditions.

```
after 6.seconds do
  val = {:id => '2', :payload => 'StartDevice'}
  g.resources[type: 'bikesnet'].cmd = val
end
```

Listing 1: An example command

The aforementioned features will illustrate the delay tolerance of the framework, by caching the commands/messages from the experimenter to the sensing devices and vice versa. In addition to this, the importance of getting the device or part of its modules into "sleep" mode will be highlighted as a prominent factor for power saving. Finally, the acquired measurements will be depicted live with the help of the OMF's visualization framework.

IV. Conclusion

In this demo paper, we demonstrate the NITOS BikesNet framework that enables experimenters to remotely control a mobile wireless sensing network constituted of sensing devices mounted on bicycles. In particular, we present an experiment which shows how the proposed framework facilitates experiment execution in our custom-made and modular devices.

V. Acknowledgements

The authors acknowledge the support of the European Commission through STREP project REDUCTION (FP7-288254) and IP project EINS (FP7-288021).

References

[1] J. A. Burke, D. Estrin, M. Hansen, A. Parker, N. Ramanathan, S. Reddy, and M. B. Srivastava, "Participatory sensing," in *Proc. 1st ACM Workshop on World-Sensor-Web: Mobile Device Centric Sensory Networks and Applications (WSW), 4th ACM Conf on Embedded Networked Sensor Systems (SenSys)*, 2006.

[2] Network Implementation Testbed using Open Source platforms. [Online]. Available: http://goo.gl/j67I6k

[3] T. Rakotoarivelo, G. Jourjon, and M. Ott, "Designing and orchestrating reproducible experiments on federated networking testbeds," *Computer Networks, Elsevier*, vol. 63, 2014.

[4] D. Stavropoulos, G. Kazdaridis, T. Korakis, D. Katsaros, and L. Tassiulas, "Demonstration of a vehicle-to-infrastructure (v2i) communication network featuring heterogeneous sensors and delay tolerant network capabilities." in *Proc. 8th Intl ICST Conf on Testbeds and Research Infrastructure. Development of Networks and Communities (TRIDENTCOM)*, 2012.

[5] V. Maglogiannis, G. Kazdaridis, D. Stavropoulos, T. Korakis, and L. Tassiulas, "Enabling mobile sensing through a dtn framework," in *Proc. 8th ACM Intl workshop on Wireless Network Testbeds, Experimental Evaluation and Characterization (WiNTECH)*, 2013.

[6] S. B. Eisenman, E. Miluzzo, N. D. Lane, R. A. Peterson, G.-S. Ahn, and A. T. Campbell, "Bikenet: A mobile sensing system for cyclist experience mapping," *ACM Transactions on Sensor Networks (TOSN)*, vol. 6, no. 1, 2009.

[7] L. Sanchez, V. Gutierrez, J. A. Galache, P. Sotres, J. R. Santana, J. Casanueva, and L. Munoz, "Smartsantander: Experimentation and service provision in the smart city," in *Proc. 16th IEEE Intl Symposium on Wireless Personal Multimedia Communications (WPMC)*, 2013.

[8] Arduino platform. [Online]. Available: http://www.arduino.cc/

[9] Teensy 3.0 dev board. [Online]. Available: http://goo.gl/BMG4XM

[10] Xbee wireless module. [Online]. Available: http://www.digi.com/xbee/

[11] Wifly - rn-171xv 802.11 b/g. [Online]. Available: http://goo.gl/p7TCC5

[12] D2523T GPS Receiver. [Online]. Available: http://goo.gl/Nkw3b6

[13] SHT11 Temp & Hum Sensor. [Online]. Available: http://goo.gl/lnzf6N

[14] Wake-on-shake Board. [Online]. Available: http://goo.gl/4VwivR