NITOS BikesNet: Enabling Mobile Sensing Experiments through the OMF Framework in a city-wide environment

Giannis Kazdaridis, Donatos Stavropoulos, Vasilis Maglogiannis,
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, vamaglog, korakis, lalis, leandros}@uth.gr

***Abstract*—In this paper we present the NITOS BikesNet platform, a city-scale mobile sensing infrastructure that relies on bicycles of volunteer users. NITOS BikesNet employs a custom-built embedded node that can be equipped with different types of sensors, and which can be easily mounted on a bicycle in order to opportunistically collect environmental and WiFi measurements in different parts of the city. Experimenters can remotely reserve and control the sensor nodes on bicycles as well as collect/visualize their measurements via the OMF/OML framework, which was extended in order to handle the intermittent connectivity and disconnected operation of the mobile nodes. We also provide a performance analysis of our node prototype in terms of sensing latency, end-to-end data transmission capability and power consumption, and report on a first experiment that was performed using NITOS BikesNet in the city of Volos, Greece.**

I. Introduction

Driven by recent technological advances, mobile devices become increasingly sophisticated and miniaturized. Tablets, smartphones and various wearable devices, with embedded cameras, microphones, accelerometers, GPS and other types of sensors, are now part of everyday life. Carried by people everywhere they go, such devices can record a wealth of data, as well as give rise to new applications. For instance, the glasses launched by Google allow users to take photos and videos on the go, just by issuing voice commands. Along the same lines, smart wristwatches or clothes can infer the user's physical activity or detect accidents.

But apart from benefiting the individual, mobile and wearable sensors and information devices can also be useful for the society as a whole. In the spirit of so-called participatory sensing [1], people may contribute to a common goal by recording and publishing information on a voluntary basis, without even knowing each other; for instance, overflowed garbage bins or street potholes can be reported as part of daily commuting activities. More generally, one can view people with their smartphones and wearable devices as a mobile ad-hoc sensing infrastructure, which can be employed, in an opportunistic or targeted way, to perform large-scale sensing tasks. Note that the coverage and/or density that can be achieved this way could very well surpass that of any planned/fixed sensing infrastructure; in particular, since many governments and local authorities cannot afford to setup, operate and maintain nation-wide or even city-scale sensor networks.

At the same time, there is a rising need for the research community but also companies (in particular SMEs) to conduct experiments and pilot deployments in the real world, as opposed to a controlled lab setting. This holds even more for Internet-of-Things and crowdsensing scenarios, where real people with real behaviors and real mobility patterns are required to evaluate different approaches under realistic conditions at a large scale. However, while a lot of work has been done on testbeds that can be used remotely by researchers to run experiments on fixed wireless sensor networks, much less has been done to support experimentation with mobile sensor networks.

In this paper, we present the NITOS BikesNet framework, which can be used to realize such a testbed at the scale of an entire city, 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. NITOS BikesNet allows external researchers to plan experiments, remotely control the operation of the mobile nodes (e.g., turn on/off individual sensors, or set the sensor sampling period), and retrieve, process and visualize their measurements. The management of the sensor nodes and the data produced by them is done via the cOntrol and Management Framework (OMF) [2] and the OMF Measurement Library (OML) [3], 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 largely transparent fashion: on the one hand, each node saves its measurements on local stable storage, and incrementally uploads data to the NITOS server when it has Internet connectivity; on the other hand, the commands issued by the experimenter are queued on the NITOS server, and are forwarded to the sensor nodes when they contact the server.

The main contributions of this paper are: (i) we identify the most important requirements for a testbed that allows experimentation with mobile sensor nodes; (ii) we describe the extensions made to the OMF framework in order to deal with mobility issues; (iii) we describe the implementation of our sensor node prototype; (iv) we analyze the performance of the node in terms of sensing latency, end-to-end data transfer capability and power consumption; and (v) we present indicative results from a first experiment that was performed using our platform. We believe that at least some aspects of our work have wider applicability, and can inspire the design or enhancement of other testbeds.

The rest of the paper is organized as follows. Related work is presented in Section II. Section III lists the most important requirements for a testbed that allows experimentation with mobile sensor nodes, along with the choices we made for the NITOS BikesNet platform. Section IV presents the NITOS BikesNet architecture, the extensions made to OMF to deal with the mobility of the sensor nodes, and the implementation of our sensor node prototype. Section V discusses the performance of the sensor node, while Section VI presents an experiment that was realized using NITOS BikesNet in the city of Volos. Finally, Section VII concludes the paper and outlines directions for future work.

II. Related Work

Bicycles related implementations: To the best of our knowledge, the only system that is similar to our work is [6]. It is based on the Moteiv Tmote Invent platform, and uses a variety of sensors to measure environmental parameters as well as to record data about the performance/fitness of the cyclists themselves. Unlike NITOS BikesNet, the system was not designed as an open testbed that allows external researchers to perform experiments of their own. Also, the sensing devices mounted on the bicycles are not easily customizable, whereas the NITOS mobile sensor node can be equipped with different sensors according to the application's requirements. Another work that focuses on bicycles is presented in [7]. In this case, cyclists use their mobile phone to record and share the paths traveled and ride statistics, which in turn can be used to infer the difficulty and noisiness of the various routes. However, this is a dedicated application for the bicycle owners themselves rather than a testbed infrastructure for running experiments that exploit the sensing capabilities of other people's bicycles. In addition, for all practical purposes, one is limited to whichever sensors are available on smartphones.

Sensing infrastructures/frameworks: An exemplary case of a city-scale wireless sensor network testbed is SmartSantander [8], which provides access to numerous fixed and some mobile sensor nodes deployed in the city of Santander. The mobile nodes are based on commercial sensing devices installed on public transport buses and municipality vehicles, and are mainly used to perform air quality measurements throughout the city. Moreover, the node control plane is implemented over GPRS and assumes practically constant connectivity, as opposed to NITOS BikesNet that supports an asynchronous transfer of both control commands and sensor measurements. The testbed management functionality of SmartSantander is based on an adapted version of WISEBED [9], which also supports the reprogramming of sensor nodes over the air. Still, in [8], the creators of SmartSantander acknowledge the importance of adopting a widely used experiment management framework such as OMF, which is embraced by the majority of the FIRE initiative [10] projects. As another example of a city-scale mobile sensor network, [11] describes a system developed for the city of Zurich, Switzerland, where trams carry devices that sense air pollution. However, this is an application-specific system, which uses custom node management software that does not offer any experimentation support to third parties. Another difference with NITOS BikesNet is that the routes and time schedules of the mobile sensor nodes are known in advance. Previous work on the integration of OMF with sensor resources to implement structural health monitoring of bridges [12], [13] extended the framework to support the control of fixed nodes. Unfortunately, it seems that this activity is not continued in the latest release of OMF. Also, this work did not aim to support an asynchronous and disconnected operation of OMF with mobile sensor nodes, as this is the case with NITOS BikesNet.

Smartphone-based sensing platforms: A lot of work has been done on mobile sensing platforms based on smartphones. Code-In-The-Air (CITA) [14] allows the user to write sensing tasks for his smartphone in the form of high-level scripts. These are compiled into a server-side and a mobile part, with the latter being shipped and executed on the user's phone. The interaction between the server and the phone is based on an asynchronous messaging service. The Mobile Sensor Data Engine (MOSDEN) [15] is a remote sensing framework for smartphones. It uses the concept of plug-ins to enable a flexible integration of on-board and external sensors without recompiling and/or redeploying the system. MOSDEN implements generic sensing and data storage functions directly on the smartphone, letting each device act as a server that can be used by several applications. With EasyHarvest [16], the user uploads sensing tasks written in Java on a server, which distributes them on smartphones and collects the data produced, in a transparent way. Smartphone owners control the execution of sensing tasks on their devices through a single interface, without having to repeatedly download, install and configure individual sensing applications. Like MODSEN and EasyHarvest, NITOS BikesNet is designed to engage a large number of mobile devices and can thus support city-scale sensing scenarios. While our platform does not support the shipment of application code to the mobile devices, as CITA and EasyHarvest, it lets the user remotely configure their operation. Also, the fact that NITOS BikesNet is integrated with OMF/OML simplifies the organization, controlled execution and monitoring of experiments that involve selected or all mobile sensor nodes.

III. Requirements and Implementation Choices

The goal of our work is to provide a platform for easy experimentation and prototyping with mobile sensor nodes mounted on bicycles. We envision a testbed that is open for third parties to test mobile crowdsensing ideas, methods and protocols, under real-world conditions and at city-scale. We also place great emphasis in using low-cost and energy-efficient nodes that can be easily adapted to support different sensing scenarios. Below, we list the most important requirements which we believe such a platform should satisfy, along with the choices we made for NITOS BikesNet.

Experimentation capability: The user should be able to plan, execute, monitor and control an experiment, from a remote location. This includes the reservation of (perhaps specific) mobile sensor nodes for a timespan, the parameterization of the sensing task, as well as the ability to record, retrieve, analyze and visualize the measurements taken. NITOS BikesNet builds on top of the OMF/OML framework, which is widely used to manage experiments on top of different networking testbeds all over the world. It also provides easy-to-use data visualization tools.

Remote configuration: The sensor node should work right out of the box, without the bicycle owner having to configure it in any way. The setting of the configuration parameters that drive the operation of the mobile sensor node should be done in a transparent fashion, without requiring physical access. NITOS BikesNet lets the user configure the mobile nodes according to the needs of the experiment, via OMF commands, at any point in time (also during the experiment).

Disconnected operation: The sensor node mounted on the bicycle is unlikely to have constant or reliable network connectivity. On the contrary, its connection to the Internet will most likely be sporadic and short-lived, e.g. via the open access points encountered in the city. NITOS BikesNet employs delay-tolerant communication techniques to let the transfer of commands (to nodes) and measurements (from nodes) occur in an incremental and asynchronous way, whenever nodes connect to the server.

Support for different networking technologies: WiFi is currently the most popular choice for ad-hoc wireless networking in a city, due to the abundance of open access points. Still, it is desirable to let researchers experiment with different networking technologies, either for the communication between the sensor node with the server, or between nodes themselves in the spirit of vehicle-to-vehicle communications. The NITOS BikesNet sensor node is designed to enable simultaneous usage of two separate network interfaces. Currently these can be either WiFi or ZigBee (the respective hardware modules can be plugged/unplugged, at will); we also provide a custom ZigBee access point through which nodes can connect to the Internet. Other technologies, such as Bluetooth and Cellular, could be supported in a similar way; this is future work.

Low-cost: The cost of the sensor node is crucial, especially if one targets large-scale deployments. Also, having low-cost nodes significantly simplifies their replacement, e.g., in case they brake (due to abrupt shocks/falls of the bicycle), or get stolen (if people forget to remove them when leaving their bicycle unattended). NITOS BikesNet employs a custom-build sensor node that costs less than €100 in total. The cost could drop further, if one decides to go for a mass production. Also, the installation of nodes on bicycles does not require any technical skills and has zero cost.

Low-power: The mobile sensor node should be able to operate for a longer period of time on batteries. Asking the bicycle owner to recharge the node once a day or change batteries every week, could already be quite annoying. This is even more so if people are supposed to be part of the platform for a long period (e.g., a year). The NITOS BikesNet node is lightweight, using components with low-power characteristics and sleep modes; it is also possible to completely power-off individual components. Currently, without any fancy optimizations, and under reasonable assumptions, our sensor node can operate autonomously for about a month.

Small size: The mobile sensor node should be small in order to be easily mounted on the bicycle without blocking the cyclists' moves. The NITOS BikesNet node is designed to fit underneath the bike saddle, where it is also well-protected against falls and weather conditions (rain). It can be attached to and detached from a bicycle in a few seconds.

Extensibility: It is impossible for a single device to incorporate all the sensors that could be required by different experiments. Ideally, the sensor node should be extensible, making it possible to add new sensors that are not available in the standard package. The NITOS BikesNet node is built in a modular way, so that existing sensors can be replaced with new ones, almost in a plug-and-play fashion. The same applies for the device firmware, which can be augmented with the appropriate device drivers. However, the respective effort and financial costs would have to be covered by the experimenter or some source of funding.

In the next section, we discuss the NITOS BikesNet platform in more detail, focusing on our mobile sensor node prototype and the extensions made to the OMF/OML framework in order to support disconnected operation and opportunistic connectivity of the sensor nodes via WiFi and ZigBee.

IV. NITOS BIKESNET PLATFORM

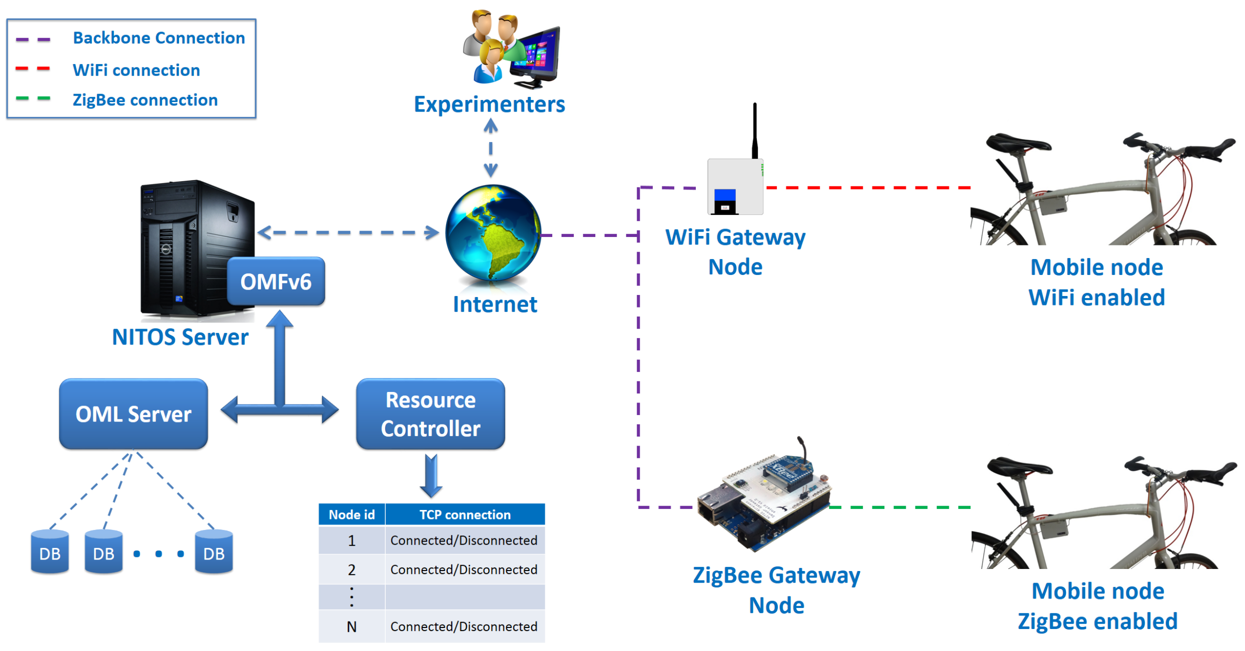
The NITOS Future Internet (FI) experimental facility [17] is one of the FIRE infrastructures, which is continuously evolving through major extensions reflecting the latest technologies and trends in the FI ecosystem. Currently, NITOS offers several testbed facilities to the research community, featuring technologies in several different areas, such as wireless networks (WiFi, WiMAX and LTE), wired networks, opportunistic networks, software-defined radios and networks, Internet of Things (IoT) and smart city infrastructures. NITOS is open and remotely accessible to any researcher who wishes to deploy and experimentally evaluate networking protocols and applications in real world settings. Users can reserve and control the respective testbed resources through the NITOS scheduler [18], which works in conjunction with the OMF management framework.

A. Overview of the OMF/OML framework

The control and management of the NITOS facility is done using the OMF open-source software. OMF was originally created in the Orbit [19] testbed, and soon became the most widely used tool for experiment control among the majority of the testbeds worldwide. Notably, OMF is the primary experiment control tool in the FIRE initiative [10] as well as in GENI [20].

In particular, OMF enables the experimenter to automate an experiment instead of setting up everything manually by logging into each node to configure/control its operation. The concept is similar to network simulators where the user describes a topology along with the applications that run during the simulation. The difference is that the topology consists of physical nodes on which OMF runs applications like a traffic generator. Also, the measurements are automatically collected with the help of the OML. The configuration and control of node operation occurs through specific properties, which are part of "formal" resource descriptions, and can be done not only at experiment setup but also during experiment runtime.

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-



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

Fig. 1: NITOS BikesNet architecture.

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.

Finally, the measurements produced by an experiment are stored by the respective RCs directly on an OML server, without any involvement of the EC. Given that several experiments can run at the same time, a separate database is maintained for each experiment. The user can inspect and retrieve the results of his experiment at any point in time.

B. NITOS BikesNet 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 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 (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. Along the same lines, when a mobile sensor node becomes available, the RC retrieves its measurements and forwards them to an OML collection point.

While the user can address/control each mobile node separately, for practical reasons, we interface all nodes via a single RC, which is responsible for keeping track of their status. This way, a single endpoint address needs to be reserved at the NITOS server, and all nodes can be pre-configured with it. Also, the EC can control several nodes via a single command to the RC, which can fan-out this command to several or all nodes. Furthermore, having only one RC conserves resources on the server without incurring significant service delays; provided that the nodes contact the server at different intervals and stay connected for a relatively short duration (which is a reasonable assumption in our case). This said, our implementation can be modified in a straightforward way to let nodes connect to different RCs for better scalability.

The internals of the RC and its interaction with the other components of the NITOS platform are shown in Fig. 2. The RC includes handlers for managing the connections with the

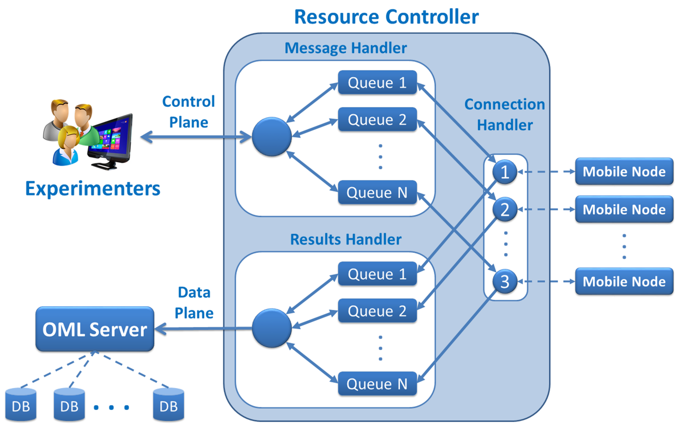


Fig. 2: Resource Controller architecture.

mobile sensor nodes, processing the control messages coming from the EC, and processing the measurements coming from the nodes. The connection handler is responsible for keeping track of the online status of the nodes, and for forwarding messages and results between the nodes and the other two handlers. The message handler basically keeps a separate queue for each node, where incoming commands are stored when the node is disconnected. Similarly, the result handler maintains a separate queue for each experiment, where it places the respective measurements received from each node, in order for them to be forwarded to the proper OML collection point.

C. 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 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.

Direction :: Message	**Comments**
N → 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 → N :: <node id>:CMD:<seq. #>:<payload> N → RC :: <node id>:CMDACK:<seq. #>	RC sends the next pending command, and waits for an acknowledgment. Can be repeated several times.
N → RC :: <node id>:RSLT:<seq. #>:<payload> RC → N :: <node id>:RSLTACK:<seq. #>	Node sends next batch of results to the RC, and waits for acknowledgment. Can be repeated several times.
N → 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).

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.

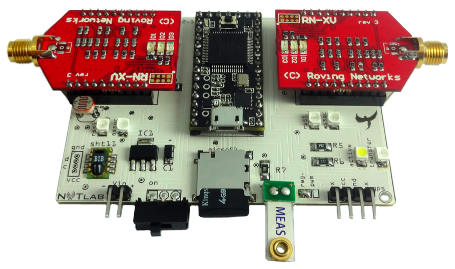
D. Developed hardware and firmware

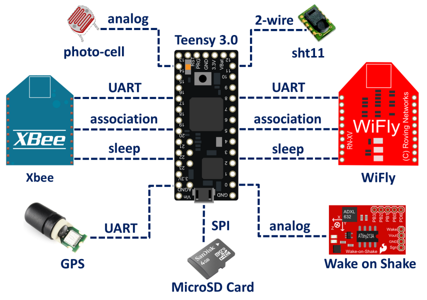
The NITOS BikesNet sensor node, depicted in Fig. 3(a), is built by combining different hardware components to create a unified solution that meets the requirements stated in Section III. Our prototype is based on an Arduino-like board and several Arduino compatible modules. We chose the Arduino platform [21] because of its open approach, great flexibility and the large number of publicly available hardware modules and software libraries. In the following, we briefly describe the components of the node and explain how they contribute to the overall node functionality. A high-level component diagram of the sensor node is shown in Fig. 3(b). We also discuss the node firmware and the custom-built ZigBee gateway.

Teensy microcontroller board: The main module of the node is a Teensy 3.0 development board [22], which features a 32-bit ARM Cortex-M4 microprocessor with 128KB of flash (program space) and 2KB of EEPROM (long-term storage) memory. The board also has 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. The latter frequency can be used during idle periods where high-processing capabilities are not required, to save energy. 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 [23], and the WiFly RN-171 module [24]. The Xbee module implements the ZigBee protocol on top of 802.15.4 in the ISM 2.4GHz


(a) Mobile sensor node



(b) Component diagram

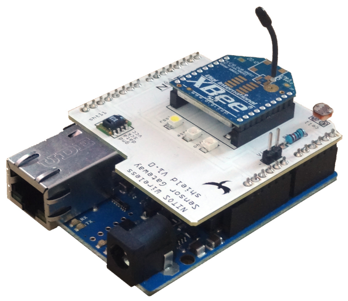

(c) ZigBee gateway

Fig. 3: Custom-built hardware of the NITOS BikesNet platform

band. The WiFly module implements the 802.11 b/g radio protocol and is compatible with commercial WiFi access points. Furthermore, it incorporates a full TCP/IP stack that can be invoked by the microcontroller of the Teensy. Both modules operate at 3.3V, thus allowing communication with Teensy without requiring an extra hardware component for logical level conversion. They also support a “command” mode, thereby allowing the Teensy to configure and control their operation in a straightforward way.

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. Alternatively, the node may feature just a WiFly module that can provide both functions, driven by the Teensy as needed.

Sensors: The node is equipped with a GPS unit and temperature, humidity and light-intensity sensors. All modules operate at 3.3V, so they can be directly connected to Teensy. In addition, all modules are powered through an individual I/O pin of Teensy, so that Teensy can turn them off for power saving.

The GPS is a D2523T receiver [25] 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, and is driven via the tinyGPS Arduino library. Teensy can dynamically turn the GPS on/off to save power.

The air temperature and humidity is measured via a sht11 digital sensor [26], connected to Teensy over a 2-wire interface. The sensor is driven using the sensirion Arduino library. Finally, a photocell sensor that consists of a light-dependent resistor is interfaced with one of Teensy’s analog I/O pins, along with an electrical resistor forming a voltage divider circuit. To acquire luminosity measurements Teensy performs an analog to digital conversion, estimates the voltage drop across the photocell and translates the obtained value into a light intensity value.

microSD: The node also features a microSD slot connected to Teensy via the SPI bus. The attached card is powered through an I/O pin of the Teensy, thus it can be turned off as the aforementioned sensor modules. 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. Moreover, the node can be put in sleep mode or be completely turned-off without data loss, which is of crucial importance in order to achieve good autonomous operation on batteries. It also ensures that measurements will not be lost in case the node resets/reboots or runs out of batteries.

Wake-on-shake: The Wake-on-Shake (WoS) [27] 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. To achieve this, the analog pin of WoS is connected to an interrupt pin of the Teensy that is pre-configured to awake Teensy in case of sensing a “HIGH” signal. 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. The software is custom-developed but also relies on several open-source Arduino libraries, mainly for controlling the peripherals.

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. Note that this approach is quite meaningful, as it is typically not useful to take many samples at exactly the same location. More importantly, the node remains asleep when it is not moving, which we expect to be the case most of the time (a bicycle is typically used only for short periods during the day). Of course, it is possible to implement more refined/clever policies, which we plan to do in the future.

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. When using a network module (WiFly or ZigBee) as a communication channel, the firmware monitors the association pin to detect that a connection to a gateway has been established. Then, it

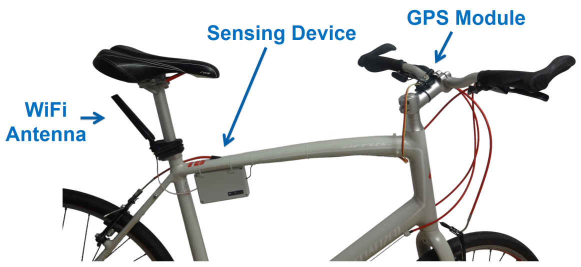


Fig. 4: NITOS mobile sensor node mounted on bicycle

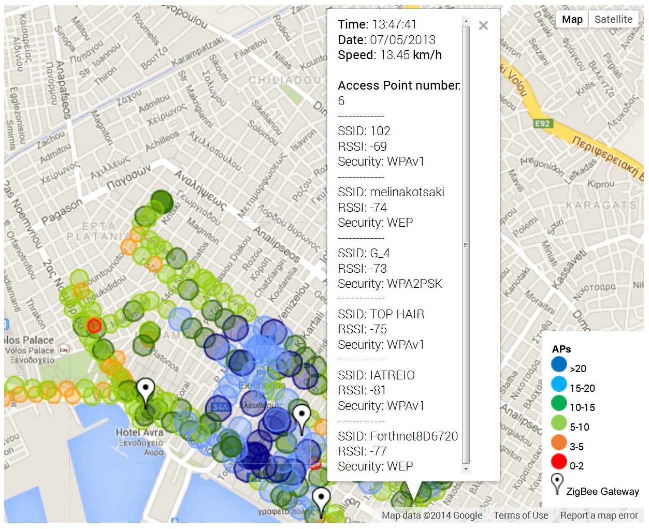
initiates a conversation with the RC (as described previously). To save power, this is done only if the node has some measurements to upload and it has not already contacted the RC recently, else the module used for communication is put in sleep mode.

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. On the other hand, an alternative option would have been to use an IP stack for resource-constrained devices, such as 6LoWPAN [28], which allows IPv6 packets to be sent and received over IEEE 802.15.4 networks. However, this would require additional development effort and would substantially increase the complexity of the firmware without providing any extra functionality in our framework.

The association with NITOS ZigBee gateways is done using a pre-configured PAN id, stored in the node's EEPROM. This can be done because all ZigBee gateways are part of our testbed. For WiFi, we obviously want the node to exploit third-party access points that are available all over the city. To this end, the firmware maintains in EEPROM a list of known APs along with the respective passwords. Like other node configuration parameters, this information can be remotely updated by issuing corresponding control commands via the EC/RC.

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. 3(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. We do not currently use the security support of XBee, but this could be easily done, if required, in the future.

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. When the association is lost, the gateway sends a "BYE" message on behalf of the node to the RC, and closes the TCP/IP connection. This proxying is transparent for the RC, which handles all TCP connections (whether through WiFly or the ZigBee gateway) in the same way.



```
Time: 13:47:41
Date: 07/05/2013
Speed: 13.45 km/h

Access Point number:
6
-------------
SSID: 102
RSSI: -69
Security: WPAv1
-------------
SSID: melinakotsaki
RSSI: -74
Security: WEP
-------------
SSID: G_4
RSSI: -73
Security: WPA2PSK
-------------
SSID: TOP HAIR
RSSI: -75
Security: WPAv1
-------------
SSID: IATREIO
RSSI: -81
Security: WPAv1
-------------
SSID: Forthnet8D6720
RSSI: -77
Security: WEP
```

Fig. 5: Measurement points with details.

Note that in principle the NITOS ZigBee gateway could use a single TCP/IP connection for all nodes; de-multiplexing is possible since each protocol message carries the node id. However, this would be meaningful only if a large number of nodes were likely to use a gateway simultaneously, which is currently not the case in our deployment (we have deployed the NITOS mobile sensor node on just a few bicycles).

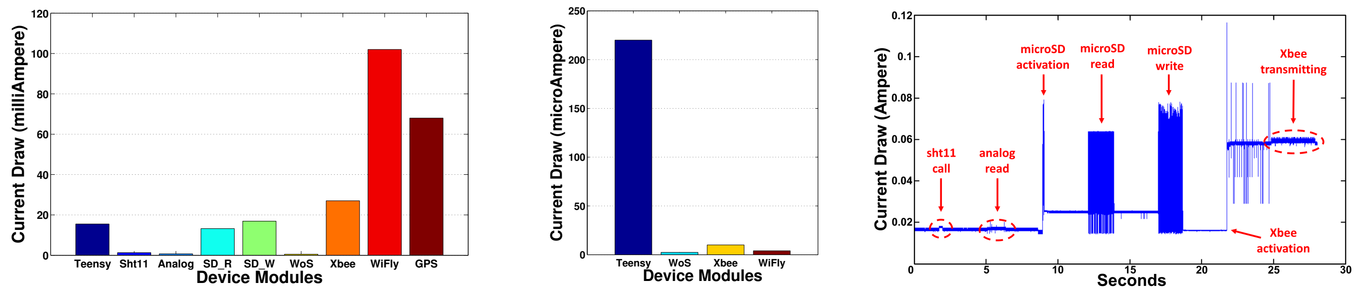
E. Installation of the NITOS sensor node

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. 4 illustrates the complete setup on one of our bicycles. Note that the most expensive parts, i.e., the node and the GPS, can be effortlessly mounted and unmounted, in a few seconds. This is important since the process may have to be repeated several times during the day, whenever leaving the bicycle unattended.

F. Visualization tools

Researchers typically need to visualize and analyze large data sets collected from their experiments, in an easy and flexible way. To this end, we have developed tools that can be used to display measurements as well as to perform different filtering and aggregation operations. This functionality is provided through the NITOS portal [29] using Google Maps [30]. Behind the scenes, we provide a mechanism that retrieves raw measurements from the corresponding experiment OML database, and performs the necessary pre-processing before feeding data into the map.

Specifically, the user can select for display specific data sets from those that were produced by the experiment, or view all collected measurements at once. He can also filter measurements based on their values and attributes (WiFi networks, environmental parameters, GPS coordinates, etc.), for instance to consider only the measurements taken in a specific region, or to omit measurements that do not satisfy a property of interest. Moreover, in case of large data sets where it is likely to have several measurements at adjacent locations, one can merge measurements that lie within the same area



(a) Active state (b) Sleep state (c) Real-time footprint of different components

Fig. 6: Power consumption of the NITOS mobile sensor node

into a single data point by supplying the aggregation radius. The maps are interactive, allowing the user to retrieve all the details of a specific data point just by clicking on it, as can be seen in Fig. 5. Of course, the user always has the option to retrieve the raw measurements from the OML database in order to perform his own data processing and visualization.

V. Performance of the Mobile Sensor Node

This section analyzes the performance of the NITOS mobile sensor node in terms of sensing latency, communication capability and power consumption. These performance figures give an idea about the potential as well as the limitations of our prototype, and can be used to guide the development of enhanced versions.

A. *Sensing latency*

We have performed several tests in order to measure the latency of each sensor used in our node prototype. The results presented next were derived after repeating each measurement a number of times to eliminate random effects and short-term fluctuations.

Data acquisition from the sht11 sensor requires 319 ms, while 230 μs are needed to perform five consecutive readings of the photo cell (the effective luminosity level is computed as the average of these values). The time for receiving GPS data over UART is 2 ms. Also, it takes about 3180 ms to invoke the WiFi scan procedure and collect the results; this time can vary depending on the number of the discovered networks due to the limited bitrate of the UART port to WiFly. The time required to store all the above measurements (with 10 WiFi access points being detected) in the microSD card is approximately 13 ms. In total, the time required to perform a full sense-and-log cycle is roughly 3514 ms. Hence the minimum sensing period, when using all onboard sensors, is about 4 seconds (the node can perform up to 15 full sensing cycles a minute)..

B. *Connectivity & transmission latency*

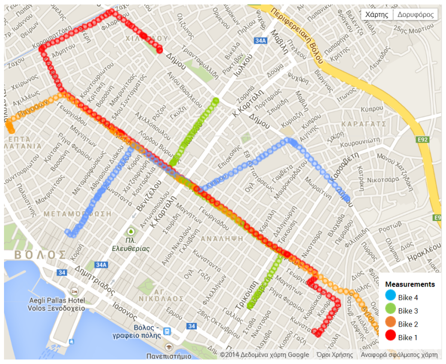
An important performance factor is the time required to connect to a gateway, and the data transfer rate that can be achieved, using a given networking technology. For Xbee, we measured a network association delay of roughly 7 seconds, whereas for WiFly the delay typically varies between 2 and 4 seconds. The rate at which data can be uploaded to server (RC) depends not only on the nominal bandwidth of the networking technology but also on the bitrate supported by the UART ports and the networking modules of the sensor node. More specifically, while the Xbee module has a physical rate of 250 Kbps, it only supports a maximum UART baud rate of 57600, which results in an effective throughput of about 46 Kbps. The WiFly module, which has a nominal physical rate of 464 Kbps and supports a much higher baud rate of 460800 over the UART, achieves a throughput of 358 Kbps.

To get a practical feeling of these numbers, let us assume the mobile sensor node has collected 100 measurements from all onboard sensors; with continuous operation and a sensing period of 10 seconds, the node would take roughly 16-17 minutes to collect these measurements. The total data size is about 91 KBytes. The time needed to upload this data on the NITOS server is close to 16 seconds over Xbee, and only about 2 seconds over WiFly. Also, if one takes into account the respective network association delays, the total amount of time for completing the data transfer is roughly 23 seconds and 6 seconds for Xbee and WiFly, respectively. Note that the delay for WiFly is quite acceptable, even for performing data upload on the move, since bicycles do not move very fast in a city and thus are likely to remain in range of an access point for some time. Clearly, this is not the case for Xbee. However, the NITOS ZigBee gateways are installed only at our office building and a few homes, where bicycles typically stay for hours, giving nodes more than enough time to upload the data collected. We are currently considering different options to achieve better upload performance, e.g., by employing the modern Digi S6B WiFi module [31], which nominally supports a rate of 1Mbps over UART and up to 6 Mbps over SPI.

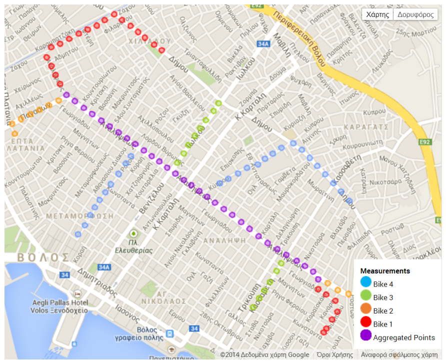
C. *Power consumption*

To measure the instantaneous power consumption of the components used in our prototype device, we follow a widely adopted measurement methodology whereby a high-precision, low impedance current-shunt resistor is placed in series with the power source and the power supply pin of the component to be measured. We use custom-built hardware developed in previous work [32], which enables online and accurate monitoring of the voltage drop across the resistor at the high sampling rate of 63 KHz via the prototype NITOS ACM card.

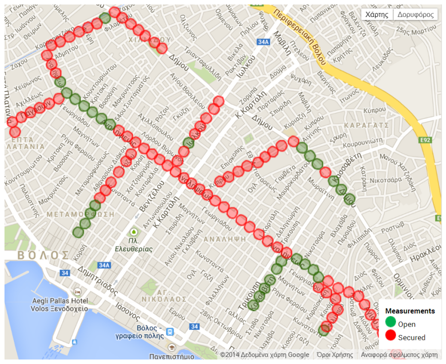
The current draw of the components used in the NITOS mobile sensor node, when in active mode, is shown in Fig. 6(a). As expected, the most expensive ones are the GPS and the two wireless modules. We note that the acquisition of temperature, humidity and luminosity measurements as well as the microSD operations are performed once per sensing period and last only for a short period of time. This implies that their power consumption does not highly affect the node's average power consumption expenditure. Fig. 6(b) shows the current draw of the components that can be put in sleep mode, when in this state: roughly 220 μA for Teensy, 2.35 μA for WoS, 10



(a) Data collected from each bicycle.



(b) Aggregated results from all bicycles.



(c) Open/secured WiFi networks discovered.

Fig. 7: Visualization of the WiFi discovery experiment results

μA for Xbee and 4 μA for WiFly. The difference compared to the active mode is huge (μA vs. mA). This clearly indicates that significant energy savings can be achieved by putting these components in sleep mode as often as possible.

Finally, Fig. 6(c) plots the current draw of the sensor node while various components are activated to perform typical sensing, storage and data transmission tasks (in this case, the node communicates with the RC via Xbee). We note that (for visualization purposes) the footprints shown here correspond to an artificially prolonged operation of the respective components, not to a typical sensing cycle. Based on repeated measurements in active mode (when the bicycle is moving), involving all the onboard sensors and with a sensing/logging period of 10 seconds, we estimate that the device drains 256 mA on average. When put to sleep (the bicycle does not move) the node requires negligible energy. We have equipped the node with 3 type AAA batteries giving a total capacity of 6600 mAh in order to achieve more than 25 hours of continuous operation. Assuming a bicycle is on the move for less than 40 minutes a day, this results in a lifetime of one month; after that, the bicycle owner has to change the batteries of the node.

VI. A Use Case / Experimentation Scenario

For the time being, we have deployed NITOS mobile sensor nodes on a few bicycles that belong to members of NITlab [33], who volunteered to try out the NITOS BikesNet platform. We have performed a number of small experiments to test and debug the system. Here, we briefly report on one of those experiments.

The objective of this particular use case is to discover and report the available WiFi networks in the city of Volos. In this case, we employed four bicycles. Each cyclist was instructed to follow a different route (the routes where partly overlapping). The sensor nodes on the bicycles were equipped with a WiFly module used to sense WiFi networks, and an Xbee module for the communication with the RC. Via OMF, the nodes were configured to perform a sensing operation every 5 seconds (as long as they were moving). The collected measurements were uploaded to the NITOS server at the end of the route, via a ZigBee gateway.

Fig. 7(a) depicts the individual measurements collected by each bicycle. All data points on the map are clickable and a pop-up box appears containing information concerning the nearby WiFi networks. An aggregated view of this data is shown in Fig. 7(b), where adjacent measurements from different bicycles are merged into a single data point in the map. Finally, Fig. 7(c) shows the result of filtering this data in order to distinguish between open and secured networks.

Bike id	Samples	Duration	Distance	Avg. speed	Data Size
1	136	11':15"	3.7 Km	19.7 Km/h	106 kB
2	82	6':46"	1.7 Km	15.1 Km/h	67,7 kB
3	127	10':28"	3 Km	17.2 Km/h	97,9 kB
4	102	8':25"	2.2 Km	15.7 Km/h	84,7 kB

TABLE II: Statistics of the WiFi-scan experiment.

Table II lists, for each bicycle, the number of samples collected, the duration of the course, the distance covered, the average speed and the total amount of data generated (all sensors were active during the experiment, not just the WiFi sensor). Based on these numbers, one can estimate that the mobile nodes took a measurement every 20-27 meters, and generated data at rates between 1,28 and 1,36 Kbps. By extrapolating these results, one can also estimate that after one hour of biking (with the same sensing frequency as before) each node would produce about 612 kB of data. Uploading this data on the server via a ZigBee gateway would then take a bit less than 2 minutes. This shows that our platform can support high-frequency and long-term sensing tasks, without the respective data uploads being prohibitively expensive.

VII. Conclusion and Future Work

We presented the NITOS BikesNet platform, which can be used to realize a city-scale mobile sensor testbed, based on nodes that are mounted on bicycles. The experimenter can control these nodes via the OMF/OML framework, in a convenient way. Such a testbed could stimulate research and service provisioning in the areas of mobile ad-hoc sensing, participatory and crowdsensing applications. We also believe that our work can provide valuable insights to researchers working in similar testbed platforms.

Our future plans include the extension of the NITOS sensor node to support additional types of sensors, e.g., for measuring noise, air pollution, and performing spectral scans of the wireless bands. We also wish to test alternative wireless access technologies for the communication between the mobile nodes and the NITOS server, such as Cellular or Bluetooth, as well as to investigate combined usage scenarios. Another item of high importance is the implementation and evaluation of smarter power management policies at the firmware level, to further increase the autonomy of the mobile nodes.

We are also interested in exploring the co-existence of the

NITOS mobile sensor node with the smartphones of bicycle owners [34]. The latter could play the role of a gateway to be used for urgent interactions with the NITOS server, e.g., high-priority commands or crucial event notifications. It would also be possible to exploit the smartphone's GPS and accelerometers, as well as its extremely powerful CPU to offload heavyweight processing tasks.

Finally, we are considering more application scenarios that could be supported using our platform, like environmental monitoring, the detection of potholes on roads, or even inferring traffic jams to propose alternative routes. We also intend to make the NITOS BikesNet platform known to the wider community of Volos, in order to attract volunteer cyclists outside the University microcosm but also to inspire people to come up with their own application ideas.

VIII. Acknowledgements

The authors acknowledge the support of the European Commission through STREP project REDUCTION (FP7-288254) and IP project EINS (FP7-288021). Moreover, the authors would like to thank NITlab members for their valuable help during the use case realization.

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

[3] O. Mehani, G. Jourjon, T. Rakotoarivelo, and M. Ott, "An instrumentation framework for the critical task of measurement collection in the future internet," *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] S. Reddy, K. Shilton, G. Denisov, C. Cenizal, D. Estrin, and M. Srivastava, "Biketastic: sensing and mapping for better biking," in *Proc. 28th ACM SIGCHI Conf on Human Factors in Computing Systems*, 2010.

[8] 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.

[9] FP7-ICT-2007.1.6-224460. Project WISEBED. http://www.wisebed.eu.

[10] FIRE: Future Internet Research and Experimentation initiative official website. [Online]. Available: http://www.ict-fire.eu/

[11] D. Hasenfratz, O. Saukh, C. Walser, C. Hueglin, M. Fierz, and L. Thiele, "Pushing the spatio-temporal resolution limit of urban air pollution maps," in *Proc. 12th Intl Conf on Pervasive Computing and Communications (PerCom)*, 2014.

[12] Y. Tselishchev and A. Boulis, "Wireless sensor network tesbed for structural health monitoring of bridges," in *Proc. 8th IEEE Conf Sensors*, 2009.

[13] A. Boulis, R. Berriman, S. Attar, and Y. Tselishchev, "A wireless sensor network test-bed for structural health monitoring of bridges," in *Proc. 36th IEEE Conf on Local Computer Networks (LCN)*, 2011.

[14] L. Ravindranath, A. Thiagarajan, H. Balakrishnan, and S. Madden, "Code in the air: Simplifying sensing and coordination tasks on smartphones," in *Proc. 12th ACM Workshop on Mobile Computing Systems & Applications (HotMobile)*, 2012.

[15] P. P. Jayaraman, C. Perera, D. Georgakopoulos, and A. Zaslavsky, "Efcient opportunistic sensing using mobile collaborative platform mosden," in *Proc. 9th IEEE Intl Conf on Collaborative Computing: Networking, Applications and Worksharing (CollaborateCom)*, 2013.

[16] M. Katsomallos and S. Lalis, "Easyharvest: Supporting the deployment and management of sensing applications on smartphones," in *Proc. 1st IEEE Workshop on Crowdsensing Methods, Techniques, and Applications (Crowdsensing), 12th IEEE Conf on Pervasive Computing and Communications (PerCom)*, 2014.

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

[18] A.-C. Anadiotis, A. Apostolaras, D. Syrivelis, T. Korakis, L. Tassiulas, L. Rodriguez, and M. Ott, "A new slicing scheme for efficient use of wireless testbeds," in *Proc. 4th ACM Intl Workshop on Wireless Network Testbeds, Experimental Evaluation and Characterization (WiNTECH)*, 2009.

[19] D. Raychaudhuri, I. Seskar, M. Ott, S. Ganu, K. Ramachandran, H. Kremo, R. Siracusa, H. Liu, and M. Singh, "Overview of the orbit radio grid testbed for evaluation of next-generation wireless network protocols," in *Proc. 3rd IEEE Conf on Wireless Communications and Networking (WCNC)*, 2005.

[20] M. Berman, J. S. Chase, L. Landweber, A. Nakao, M. Ott, D. Raychaudhuri, R. Ricci, and I. Seskar, "Geni: A federated testbed for innovative network experiments," *Computer Networks, Elsevier*, vol. 63, 2014.

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

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

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

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

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

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

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

[28] G. Mulligan, "The 6lowpan architecture," in *Proc. 4th ACM workshop on Embedded networked sensors*, 2007.

[29] NITOS visualization tool. [Online]. Available: http://goo.gl/cojA8x

[30] Google Maps API. [Online]. Available: http://goo.gl/aqsBzH

[31] Digi S6B WiFi Interface. [Online]. Available: http://goo.gl/YuT5Ta

[32] S. Keranidis, G. Kazdaridis, V. Passas, T. Korakis, I. Koutsopoulos, and L. Tassiulas, "Online Energy Consumption Monitoring of Wireless Testbed Infrastructure Through the NITOS EMF Framework," in *Proc. 8th ACM Intl Workshop on Wireless Network Testbeds, Experimental Evaluation and Characterization (WiNTECH)*, 2013.

[33] Network Implementation Testbed Laboratory. [Online]. Available: http://nitlab.inf.uth.gr

[34] Z. Ruan, E.-H. Ngai, and J. Liu, "Wireless sensor network deployment in mobile phones assisted environment," in *Proc. 18th Intl Workshop on Quality of Service (IWQoS)*, 2010.