Available online at www.sciencedirect.com

SciVerse ScienceDirect

Procedia Computer Science 00 (2015) 000–000

Procedia Computer Science

www.elsevier.com/locate/procedia

The 6th International Conference on Ambient Systems, Networks and Technologies (ANT 2015)

The Programmable City

Pedro M. N. Martins*, Julie A. McCann

Department of Computing, Imperial College London, 180 Queen's Gate, London SW7 2AZ, UK

Abstract

The worldwide proliferation of mobile connected devices has brought about a revolution in the way we live, and will inevitably guide the way in which we design the cities of the future. However, designing city-wide systems poses a new set of challenges in terms of scale, manageability and citizen involvement. Solving these challenges is crucial to making sure that the vision of a programmable Internet of Things (IoT) becomes reality. In this article we will analyse these issues and present a novel programming approach to designing scalable systems for the Internet of Things, with an emphasis on smart city applications, that addresses these issues.

Peer-review under responsibility of the Conference Program Chairs.

Keywords: Ubiquitous Computing; Programming Languages; Macroprogramming; Internet of Things

1. Introduction

The ubiquitous computing revolution has brought with it profound and widespread changes in our daily lives. It has affected how we interact with others, the role of memory when we can constantly access digital storage, how products are marketed and sold effectively through new channels, how we perceive and elect our leaders, amongst many other aspects. These changes all emerge from combinations of embedded technologies and the availability of mobile connected devices, and the services being offered to improve our individual lives. This emergent revolution poses significant questions regarding the way in which we design the spaces that we inhabit. Such questions pertain both to how we can optimise existing services so they function at their best given our new expectations, as well as how to integrate these new models of individuality, interaction, commerce, etc. into the spaces and services that the old models used to occupy.

The design of urban environments along these lines relate to smart city initiatives and are already starting to happen worldwide, in pilot programs for instance in Santander[11] Singapore[10] and London[2]. On the commercial side, we have also seen the appearance of companies such as StreetLine Networks[12] and Worldsensing[14], providing practical solutions with current technologies. A key common factor in all these enterprises is the need to understand the city better, in terms of how individuals use the city and how the city is affected by them. For example, we can think of sensing the movement of people through the city to build population density models and thus know what areas attract

* Corresponding author. Tel.: +44 (0) 20 7594 8298 ; fax: +44 (0) 20 7594 8932
E-mail address: pm1108@imperial.ac.uk

Peer-review under responsibility of the Conference Program Chairs.

attention and in what circumstances[27]. Another example is where we sense the impact on the environment that the city services are having, for example by monitoring hot spots of air pollution and using this information to guide traffic policies and road design and investment. One example of an existing project along these lines is the London Air Quality Network, collecting air quality information in and around Greater London[6]. Sensing is thus key to inform the design of future systems and to enable automation of services within the city infrastructure. Unsurprisingly then, the first stage in all the Smart City initiatives has been to instrument the city with sensing devices.

The vision is thus to have widespread sensing of city conditions, and using this to guide infrastructure, policies and services. However, one must be careful in the way in which one instruments the city and not lose track of long term consequences. For instance, there are complexities and costs associated with the infrastructures (physical and communications) that will be deployed. However, this is nothing compared with the potential costs of maintaining such infrastructures. Further, if sensing truly becomes ubiquitous, bringing with it an explosion of devices communicating data, then existing network infrastructures will surely become saturated. This has been foreseen by the research community, and hard limits have been established some time ago[16].

The idea of programming a network as a whole has been around for a while in the WSN area, and is termed macroprogramming there. This idea is used for instance for distributed data query and processing, for instance in tinyDB[21] and Regiment[24]. While the concepts are similar, macroprogramming in WSN focuses on the aspect of computation that pertains to data collection sensor networks, i.e. querying sensed data for a network as a whole and performing operations on them. Moreover, traditionally in WSN research the focus is on heavily constrained devices. As a result, most of the existing solutions have restrictions on the types of computations that can be performed to make the problem more tractable. Whilst we also tackle the issue of data query when specifying the sources to be used for an application, in this article we propose a mechanism for extensions to a general purpose language, whereby the exact location in the network where to run code is determined based on an external specification of requirements and data queries. These applications handle the whole spectrum of computation, including sensing, processing and actuation duties. Moreover, we can run any program that can be run on the existing operating system. In research in mobile computation and sensing offloading[18,26] we see a similar move towards executing equivalent types of tasks in both fixed and mobile nodes. This requires the same features and abstractions as in our work in conditionally deploying computation throughout the (cloud to edge) network of heterogenous nodes, and we were inspired by this body of work.

2. Scaling IoT applications

The two major ways to combat this saturation are to try to decrease the amounts of data that are being transmitted and to effect some form of network off-loading. In practical instrumentation architectures nowadays, sensor data is typically relayed to a central sink, or cloud storage, for processing and archiving[23,25]. However, with widespread instrumentation of the city, not all of this data is equally as valuable, and some of it has transient value only. For instance, we might want to sense water pressure in a pipe with a very fine-grained resolution in order to detect a leak and then actuate a valve controlling the pipe as soon as possible to avoid unnecessary water damage. In normal situations, most of these samples will be highly redundant, and we do not need to store all of them for auditing.

In recent years, there has been a research movement to do in-network processing in sensor networks[17], to optimise their functioning. Indeed, in the previous example, if we perform the valve actuation calculations inside the network we are able to not only reduce the delay in actuating the valve, but also decrease the amount of information that is communicated to the cloud, freeing the communication medium. We believe this is essential to allow the city to be further instrumented and automated without interfering with existing services. Complementary to the in-networking approach is network off-loading. Here we minimise the communications range of the devices and instead of building ridged routing trees to multi-hop sensed data around or have each device send data directly to its sinks, we empower mobile devices to collect the data and relay it. This might utilise the physical movement of the device to carry out the communication (in data mules for example) or off-load the traditional network by using ‘free’ communications such as Bluetooth or Wifi-Direct to multi-hop data between mobile devices or piggyback sensed data over other regular communications[28].

So we can now think of a continuum where data is processed and relayed (also a form of processing) from source devices via edge devices to the cloud. The question is where do we carry out processing, with what nodes and how? In

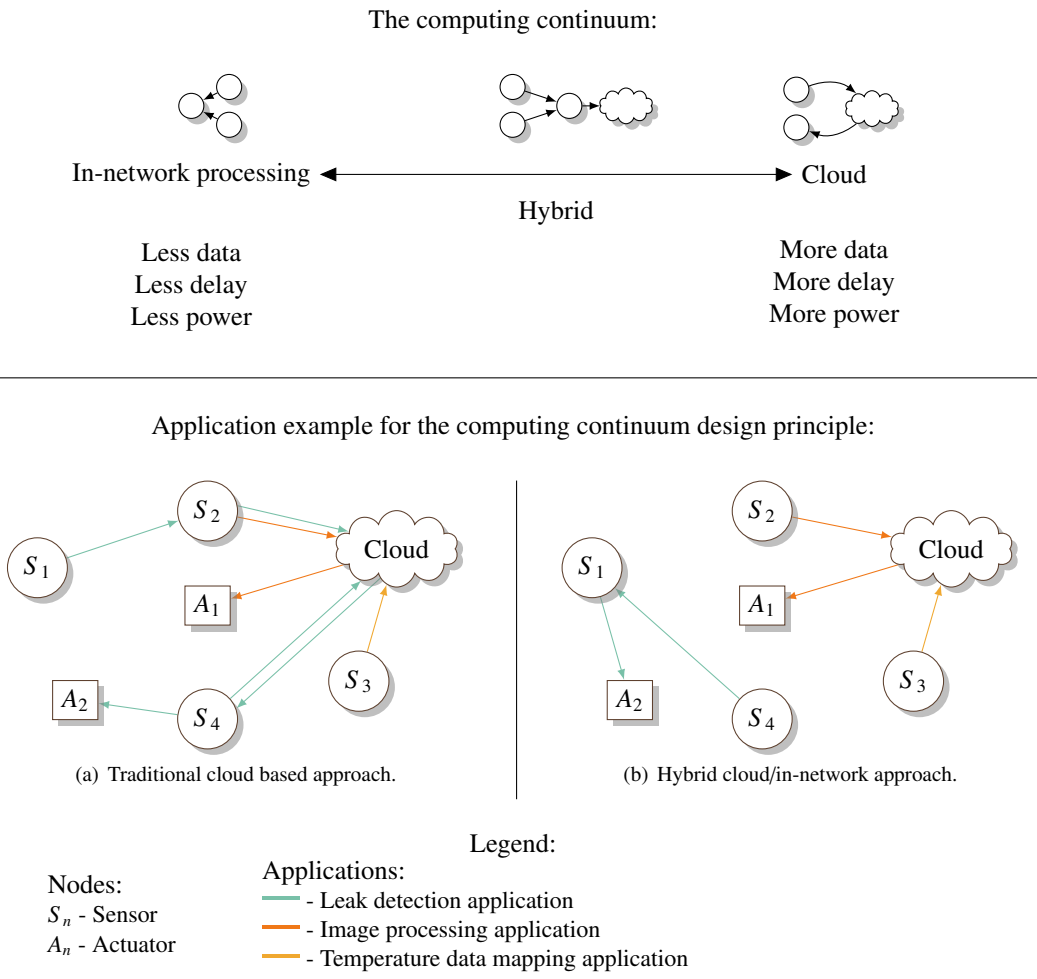

Fig. 1. The computing continuum as a design principle for IoT applications.

Figure 1 we can see an example network comprised of several sensors and actuators, as well as some cloud services. In this network we are running three different services, affecting different subsets of the nodes. The first application is the leak detection application from before, and uses data from sensors S_1 and S_4 to affect the state of the valve actuator A_2. The second application is an image processing application and uses data from sensor S_2 to guide the operation of actuator A_1. The third application, visualising temperature sensor data on a map, merely requires the output from sensor S_3. On the left hand side we see a diagram of data transfer when all the data is being transferred to the cloud for analysis and processing, before being used to affect the actuators. This has several negative consequences for the performance of the network. First of all, there is plenty of data that is being relayed via nodes that are not involved in a particular application, leading to communication congestion in those nodes (for instance S_2 is relaying data from two different applications, and is not involved in one of them). Secondly, we can imagine that the delay between reading sensor data from S_1 and actuating A_2 will be quite high, as the data needs to traverse the whole network and be processed in the cloud. Since leak detection is time-sensitive, as we explained before, this could be disastrous. On the right hand side we see a more balanced distribution, where we have eliminated some of the congestion by doing the computations for the leak detection application within the network. We have eliminated delay for this application, and reduced congestion overall. We have not however changed the operation of the image processing application, as it requires more processing power than would be available in edge nodes. Thus, where to place a particular module of computation depends on the requirements for it, e.g. in terms of timeliness, redundancy and accuracy.

We believe such hybrid approaches will be essential to allow smart city applications to both scale and be maintainable and we should start considering these problems when designing development environments for them. In this way, we should think of the whole network, from edge to cloud, as a programmable device and design technologies to facilitate this view when designing and deploying applications. These technologies will have to take into account requirements that are essential for these types of applications. For instance, if we are using shared infrastructure to build smart city applications, there are fundamental problems in terms of security and privacy that need to be resolved. Moreover, if nodes within the network are going to perform data processing for various stakeholders, we need to be able to ensure isolation between these various applications. Techniques need to be in place to efficiently and fairly utilise the shared networked resources in order to be able to deliver necessary data in a timely manner. Some form of registry needs to be provided in order to be able to discover resources and define these applications. Finally, in terms of city applications, we need to be able to support applications that could have significant real world consequences, in terms of property damage, such as the pipe leak example, or even lives, if we think about automating the healthcare systems. Thus, we need to be able to provide strong guarantees in terms of the application's behaviour when needed.

3. Virtualisation on the edge

This paradigm of computation brings with it several challenges pertaining to application development and deployment. In the previous setup, we motivate nodes running multiple applications that might belong to different stakeholders. This is not a new problem in the field of cloud computing, where computing power is shared between different users, and the users have no control over where their application is running. These applications need to be run in isolation. The current solution for this problem in the cloud computing sector is virtualisation. Indeed, through the usage of virtual machines in our case, applications can be deployed on any physical node, and transferred without issue. Each individual node can run several applications, while every particular application views the physical hardware as its own and is not affected by the operation of other applications. However, the virtualisation concept is not trivial to extend to the edge of the network, where devices are low powered, have low processing capabilities and have stringent battery constraints. One solution we have explored for this problem is to use lighter forms of virtualisation, such as operating system-level virtualisation. Operating system-level virtualisation is a technique whereby all the application instances, called containers, share the kernel of the operating system. Thus, the kernel is responsible for providing isolation between these user-space instances, for instance in terms of file system, access to devices and delimiting CPU quotas. This approach to virtualisation has the advantage of not requiring several full instances of the operating system to be running, thus providing sub-second start-up times and being less CPU and memory intensive on the host. The disadvantage is that all the applications need to be running on the same operating system. This approach to virtualisation was added to the linux kernel by the linux containers project[4] and support for easier development, deployment and instance management is offered by the docker platform[1]. It should be noted however, that this approach can extend to any operating system that supports isolation of processes in the same manner.

Another challenge is that the decision regarding where to process data needs to account for the heterogeneous processing capabilities available to the continuum of devices/middle boxes/systems, and the fact that some nodes may be battery powered, and we would like to maximise their lifetime. It is tempting to then adapt solutions reminiscent of desktop computing, whereby a runtime is used on all nodes to abstract away the underlying hardware. This is the approach taken in platforms such as Fog Computing[15]. However, we believe that given that the nodes are always online and deployment can be initiated remotely and at any time, it makes sense to instead adapt compiler based approaches where optimisation is done when the application is compiled/initially deployed. That is, we believe that in order to cope with the large heterogeneity present in our target networks we should aim to have controlled variation, rather than arbitrary homogeneity. Enforcing homogeneity on heterogeneous devices can lead to problems in understanding the behaviour of applications, as we can never know for sure which modules will be running and on which platforms they will end up on. This makes the behaviour of the system less intelligible. Nevertheless, runtime approaches also have their advantages. It is much easier to provide tools and development support once the hardware differences have been abstracted away. Moreover, runtime environments are more adaptable as the runtime itself can handle migration and determine what nodes should be running the application at any given time depending on changes in available resources, etc. However, the heterogeneity of devices and environments already poses significant difficulties to application design and development, and abstracting this away can be harmful, by

either precluding platform-specific optimisations, or hiding these in a way that makes the behaviour of the system less intelligible. Intelligibility and manageability are extremely important to make sure that application development is feasible in this complex environment. Otherwise, complex interactions involving the placement of computation and communications can be hard to foresee. For example, migration of components in a highly distributed application is non-trivial, as it has to be handled by all the components that communicate with the migrated component. Otherwise, in a time-critical application like the pipe leak example, it can happen that the system takes a long time to propagate the updated addresses for the application. This can lead to delays in the sensing-actuation loop, which are not due to the system malfunctioning, but just the difficulty of accurately predicting delays in a system that is highly dynamic by default. This link between dynamicity and lack of intelligibility motivates our choice of deployments remaining static by default and only dynamic when necessary. Static systems are easier to reason about, and thus also easier to guarantee the correctness of.

4. Application development

In order to showcase the paradigm described above we have implemented a skeletal framework and the pipe leak monitoring example we described. In this example, we have a water pipe that is being monitored for pressure, and a valve that can be actuated remotely. We have assumed that the nodes are capable of running Linux, so that we can use containers. We also delegate the handling of networking tasks to Linux. This is a reasonable assumption with the appearance of devices, for instance from National Instruments[13], or Texas Instruments[5] capable of running linux in embedded platforms. On the hobbyist side, we also have seen platforms like the Raspberry Pi[8] or the Intel Galileo[3] providing low-cost fully-featured computing nodes for use in do-it-yourself IoT applications. We will implement three concurrent applications in the system. The first one will be a straightforward dashboard presenting all the information from the water network, including historical raw sensor readings and statistics, which we named `analytics`. The second one will be the valve control application, that, based on the pressure reading, detects whether there is a leak and actuates the valve accordingly, which we named `leak`. Finally, we will also present a manual override through a web page, allowing the water company to override automatic control of the valve. This application is called `control`. A representation of the data flow in these applications can be found in Figure 2. We have designed a domain specific language in Haskell, which we called Scale, allowing us to develop these applications without thinking about programming individual devices. In this language we have a notion of abstract sensors and actuators, which are resolved when the application is deployed. In this case we have `valve`, `pressure`, `dashboard` and `external control`

In Scale we can then write code for the aforementioned applications. For example in the pipe control case we can load the specification so we can refer to the network components, query the concrete source, perform the calculation to determine whether a leak is likely and actuate the valve. The code for the three applications in the example can be found in Figure 2. The main commands that will be interpreted by the compiler to determine how to distribute computation are `Scale.querySource`, `Scale.handleSource` and `Scale.executeCommand`. As an example of this distribution, whenever the compiler finds `Scale.handleSource` in the application code, it will add code to the sensor image to periodically report the sensor readings, and to the processing node to handle these readings when they are reported, based on the MQTT publish-subscribe mechanism[7]. We assume that all devices are connected to the Internet. Security is handled through MQTT security, which is based on SSL encryption[20]. The Scale compiler that we designed will thus take the application code and produce containers to run on the edge devices (both sensing and actuator processing units, e.g. the ones associated with the pressure sensor and the valve in our example) and the cloud. We can then deploy the application by assigning physical IP addresses to each name, push the images remotely to those devices and start the containers (in our case, we used SSH).

This example showcases the model that we advocate for building IoT software. We believe the focus should be shifted from programming the individual devices and the cloud separately to thinking about the whole network, cloud to edge, as a programmable environment. An open question lies in determining what abstractions are better suited for presenting this unified view of the network, such that we do not hide details that are necessary to debug the applications and understand its behaviour. We have deployed this scheme on the sensor nodes that will form part of the London Living Labs, in Hyde Park in London. This deployment will comprise of several types of environmental monitoring nodes, sensing parameters pertaining to soil quality, water quality, air quality, light pollution and more. These nodes

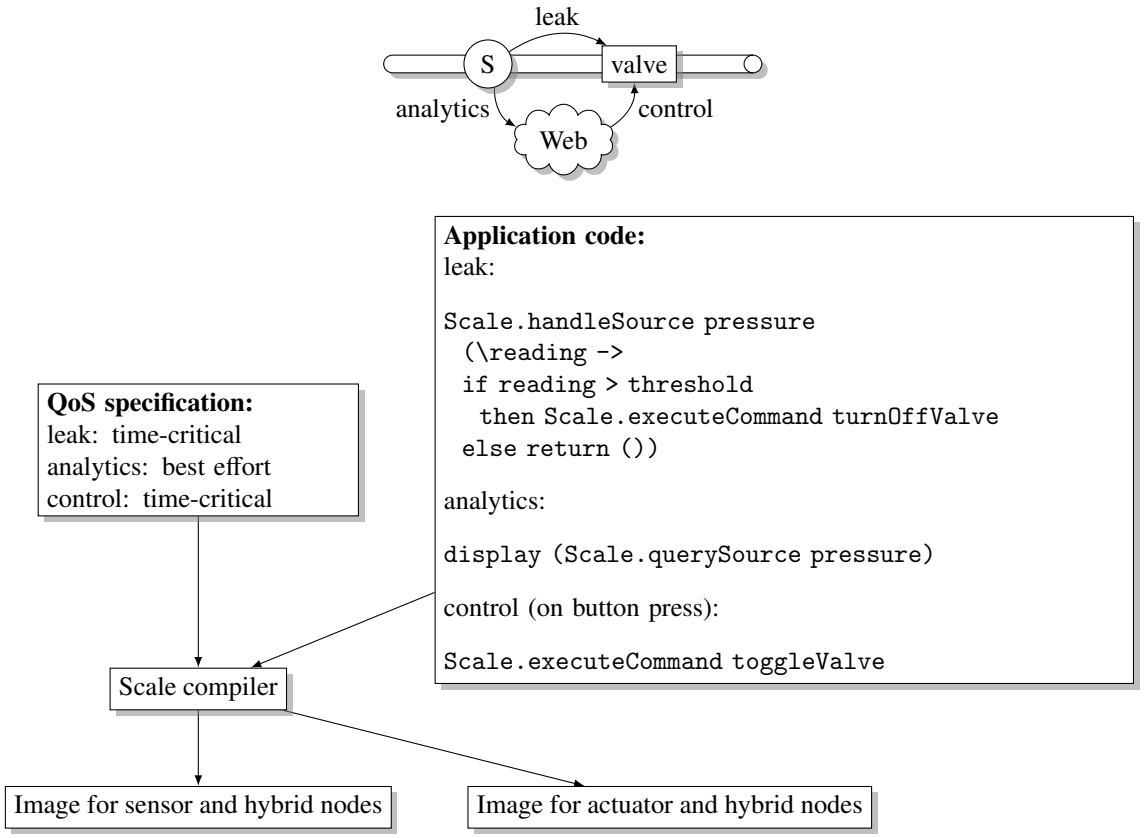

Fig. 2. The pipe leak example and operation of the Scale compiler.

use the Intel Galileo platform as mains powered gateway nodes, and the Intel Edison boards for the battery powered satellite nodes. Both of these nodes run Linux, and can be used with our framework. In the final deployment the platform will be used to allow sandboxed experimentation with the network, whilst retaining normal operation of the sensor network, delivering readings in a timely fashion. The usage of lightweight containers for edge virtualisation along with duty cycling allows us to keep the battery life of the sensor nodes practical, even though we are using fully-fledged x86 computers.

We have been exploring the idea of specifying the requirements of the application in terms of timing, redundancy, etc. so that we can then automatically determine where to deploy the code. Moreover, we might want to provide different implementations for code running in different parts of the network, for example for power optimisation reasons. It is then hard to ascertain whether the behaviour will not change by changing the specifications of the application, as we might be producing code to run on a different device. This can cause problems when devices use different internal representations for data that are not of different types. In this case, it can happen that values get interpreted erroneously, causing actuators to be effected when it is harmful for them to be. We have been exploring formal methods to make this variability more controlled. In particular, we can already use type systems to enforce the separation between two alternatives of the program (for instance one running on a cloud virtual machine and another running on an edge node), and make sure that when they are combined, this combination is well formed[22]. We have used this language to create two different versions of a sensor program, which use two different types of sensor, with different precisions. The compiler then guarantees that data produced with one sensor will never be misinterpreted as data produced by the other sensor, and that conversions occur as needed. Using current approaches, this variability is something that is usually handled with either common supertypes (potentially paired with dangerous downcasts) or tagged unions (which increase the amount of bookkeeping the programmer needs to do, usually with an associated runtime cost). Our technique allows compilers to statically guarantee that representation mismatch errors will not

happen. We can then provide stronger guarantees that critical systems will not go wrong by misinterpreting data in this way.

5. Citizen involvement and future issues

The role of citizens in the management of the city and adopted policies has definitely changed with the ubiquitous computing revolution. Adding support for citizen participation is thus paramount when designing systems for smart cities that will not alienate its inhabitants. Participatory and opportunistic sensing[19] are also becoming increasingly important due to these factors. For example, the Safecast project[9] shows how one can assemble a participatory community for doing a task that is usually fulfilled by specialised teams, in this case mapping radiation levels. In our system, from a sensing point of view, citizens contribute their own home sensing systems to be used for optimising the city infrastructure. This is achieved by having a registry of sensors, where citizens can add their devices. Moreover, if this data is made publicly available they can then use data from other relevant public devices in setting their own actuation policies in their homes. As a simple example, if they are missing an outside temperature sensor they could just use the data from the sensor of a neighbour that has been made available to control their HVAC system. A critical question when considering using other people's data is incentivisation. Our example suggests a simple model of incentivisation, where you give access to your data in return for having access to other people's data in order to optimise. However, it is more interesting to try to quantify the value of this data for each stakeholder, and trade data in a market as in our prior work[28].

Smart cities are an inevitable step forward in the design of cities, in order to integrate the technological changes that already pervade our daily lives into the spaces we inhabit. This new city-wide computing paradigm poses hitherto unseen challenges in terms of scale, heterogeneity and capacity that necessitate new development approaches and software platforms. We have proposed a unified cloud to edge solution based on compile-time containerisation of applications and automatic choice of target deployment locations according to QoS specifications for the application. This integrates the choice of where to place each software module in the in-network to cloud computing continuum, in order to create a hybrid solution for development of IoT applications, integrating both cloud and edge. This integration is transparent for the system designer, who can merely specify high-level QoS requirements. Nevertheless, intelligibility is preserved, as the deployment locations are determined statically at deployment time.

References

1. Docker. `http://docker.io`. [Online; accessed 24-Mar-2015].
2. ICRI Cities. `http://cities.io`. [Online; accessed 24-Mar-2015].
3. Intel Galileo. `http://www.intel.com/content/www/us/en/do-it-yourself/galileo-maker-quark-board.html`. [Online; accessed 24-Mar-2015].
4. Linux Containers. `http://linuxcontainers.org`. [Online; accessed 24-Mar-2015].
5. Linux Support for TI Devices. `http://www.ti.com/lsds/ti/tools-software/linux.page`. [Online; accessed 24-Mar-2015].
6. London Air Quality Network. `http://www.londonair.org.uk/`. [Online; accessed 24-Mar-2015].
7. MQTT. `http://mqtt.io`. [Online; accessed 24-Mar-2015].
8. Raspberry Pi. `http://www.raspberrypi.org/`. [Online; accessed 24-Mar-2015].
9. Safecast. `http://safecast.org`. [Online; accessed 24-Mar-2015].
10. senseable city lab. `http://senseable.mit.edu`. [Online; accessed 24-Mar-2015].
11. Smart Santander. `http://smartsantander.eu`. [Online; accessed 24-Mar-2015].
12. Streetline, Inc. `http://streetline.com`. [Online; accessed 24-Mar-2015].
13. Under the Hood of NI Linux Real-Time. `http://www.ni.com/white-paper/14626/en/`. [Online; accessed 24-Mar-2015].
14. World Sensing. `http://worldsensing.com`. [Online; accessed 24-Mar-2015].
15. Flavio Bonomi, Rodolfo Milito, Preethi Natarajan, and Jiang Zhu. Fog computing: A platform for internet of things and analytics. In Nik Bessis and Ciprian Dobre, editors, *Big Data and Internet of Things: A Roadmap for Smart Environments*, volume 546 of *Studies in Computational Intelligence*, pages 169–186. Springer International Publishing, 2014.
16. P. Gupta and P.R. Kumar. The capacity of wireless networks. *Information Theory, IEEE Transactions on*, 46(2):388–404, Mar 2000.
17. Roman Kolcun and Julie A McCann. Dragon: Data discovery and collection architecture for distributed IoT. In *Internet of Things 2014 - The 4th International Conference on the Internet of Things (IoT 2014)*, Cambridge, USA, Oct 2014.
18. Karthik Kumar, Jibang Liu, Yung-Hsiang Lu, and Bharat Bhargava. A survey of computation offloading for mobile systems. *Mobile Networks and Applications*, 18(1):129–140, 2013.

19. Nicholas D. Lane, Shane B. Eisenman, Mirco Musolesi, Emiliano Miluzzo, and Andrew T. Campbell. Urban sensing systems: Opportunistic or participatory? In *Proceedings of the 9th Workshop on Mobile Computing Systems and Applications*, HotMobile '08, pages 11–16, New York, NY, USA, 2008. ACM.
20. Dave Locke. Mq telemetry transport (mqtt) v3. 1 protocol specification. *IBM developerWorks Technical Library], available at http://www. ibm. com/developerworks/webservices/library/ws-mqtt/index. html*, 2010.
21. Samuel R. Madden, Michael J. Franklin, Joseph M. Hellerstein, and Wei Hong. Tinydb: An acquisitional query processing system for sensor networks. *ACM Trans. Database Syst.*, 30(1):122–173, March 2005.
22. Pedro M. Martins. *Context-Oriented Functional Programming*. PhD thesis, Imperial College London, June 2005.
23. Luca Mottola and Gian Pietro Picco. Programming wireless sensor networks: Fundamental concepts and state of the art. *ACM Comput. Surv.*, 43(3):19:1–19:51, April 2011.
24. Ryan Newton, Greg Morrisett, and Matt Welsh. The regiment macroprogramming system. In *Proceedings of the 6th International Conference on Information Processing in Sensor Networks*, IPSN '07, pages 489–498, New York, NY, USA, 2007. ACM.
25. Felix Jonathan Oppermann, Carlo Alberto Boano, and Kay Rmer. A decade of wireless sensing applications: Survey and taxonomy. In Habib M. Ammari, editor, *The Art of Wireless Sensor Networks*, Signals and Communication Technology, pages 11–50. Springer Berlin Heidelberg, 2014.
26. Kiran K Rachuri, Christos Efstratiou, Ilias Leontiadis, Cecilia Mascolo, and Peter J Rentfrow. Metis: Exploring mobile phone sensing offloading for efficiently supporting social sensing applications. In *Pervasive Computing and Communications (PerCom), 2013 IEEE International Conference on*, pages 85–93. IEEE, 2013.
27. Christopher Smith-Clarke, Afra Mashhadi, and Licia Capra. Poverty on the cheap: Estimating poverty maps using aggregated mobile communication networks. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, CHI '14, pages 511–520, New York, NY, USA, 2014. ACM.
28. Shusen Yang, U. Adeel, and J.A. McCann. Selfish mules: Social profit maximization in sparse sensornets using rationally-selfish human relays. *Selected Areas in Communications, IEEE Journal on*, 31(6):1124–1134, June 2013.