Self-managing Internet of Things

Danny Weyns[1(✉)], Gowri Sankar Ramachandran[2], and Ritesh Kumar Singh[1]

[1] Department of Computer Science, KU Leuven, Leuven, Belgium
danny.weyns@kuleuven.be
[2] University of Southern California, Los Angeles, USA

Abstract. Internet of Things (IoT) are in full expansion. Applications range from factory floors to smart city environments. IoT applications consist of battery powered small computing devices (motes) that communicate wirelessly and interact with the environment through sensors and actuators. A key challenge that IoT engineers face is how to manage such systems that are subject to inherent uncertainties in their operation contexts, such as interferences and dynamic traffic in the network. Often these uncertainties are difficult to predict at development time. In practice, IoT applications are therefore typically over-provisioned at deployment; however, this leads to inefficiency. In this paper, we make a case for IoT applications that manage themselves at runtime to deal with uncertainties. We contribute: (1) a set of concerns that motivate the need for self-management for IoT systems, (2) three initial approaches that illustrate the potential of realising self-managing IoT systems, and (3) a set of open challenges for future research on self-adaptation in IoT.

Keywords: Internet-of-Things · IoT · Uncertainties
Self-adaptation · Self-management

1 Introduction

Internet of Things (IoT) consist of tiny embedded and battery powered computing device (motes) that are equipped with a low-power wireless radio, sensors and actuators. These motes form networks that are capable of monitoring and controlling the physical world and thereby connecting digital processes to our physical environment. IoT applications are widely deployed in the context of industries and smart cities, see for example [5,18,21]. Typically, IoT applications require resources for computation, sensing, actuation and communication. Continuous management and maintenance of these resources is critical for accomplishing the desired stakeholder goals. This problem is particularly challenging due to the large scale nature of IoT deployments and the conditions under which they may need to operate that are often difficult to predict [17].

Consider an application example in the context of factory floor monitoring: an IoT application is deployed on the factory floor to monitor the operational conditions of the machines and production lines. In order to maintain the productivity and the efficiency of the factory floor, machines have to be operational

A M. Tjoa et al. (Eds.): SOFSEM 2018, LNCS 10706, pp. 67–84, 2018.
https://doi.org/10.1007/978-3-319-73117-9_5

24/7. To ensure this requirement, the machines are equipped with sensors that continuously monitor the temperature and the vibration profile of the machines. Whenever an abnormality is detected, the application is reconfigured to sense additional parameters of the machine. This allows fine-grained tracking of the factory environment and alarming operators in case an intervention is required. In such an application scenario, the IoT application must have capabilities to manage the resources for sensing, computation and communication. In addition, this application scenario highlights the dynamic nature of IoT applications.

Resource demands of IoT applications fluctuate during run-time due their event-driven nature [29]. Consider another application in the context of a smart building that monitors the comfort level of employees and actuates the heating when the temperature is too low, or alternatively the air condition when the temperature is too warm, and regulates the light when the light condition change. However, in the event of a fire detected in the building, the application has to be reconfigured to actuate an alarm, and stream a video to assist the fire personnel to rescue people. While monitoring the comfort level, the application requires low bandwidth, since the transmission of temperature and light reading requires few bytes of data. However, in the event of fire, the application requires high bandwidth, since the streaming of video requires at least kilo bytes of data. The ability of an IoT application to manage such dynamics autonomously and correctly is highly critical in such application scenarios.

A key underlying problem that IoT engineers face are uncertainties in the operation contexts of the applications, internal dynamics, and even changes in the requirements during operation. Often these uncertainties are difficult to predict at development time and can only be resolved during operation. To tackle these run-time uncertainties, IoT applications are typically over-provisioned at deployment. Although such an approach fulfil some of the desired application goals, e.g. the reliability, it comes at a cost of high energy consumption. Since IoT applications are battery powered, it is important to minimise its battery consumption to maximise their lifetime. With over-provisioning, IoT applications tend to be configured for worst case demands, which result in high radio use for wireless communication. According to literature, radio dominates the energy consumption in IoT application [22]. Minimising the radio usage is a major requirement for achieving a longer lifetime. Self-management frameworks that track the system and its context at runtime to resolve uncertainties during operation is essential for resource and energy constrained IoT applications.

The contributions of this paper are: (1) a set of concerns that motivate why we need self-management for IoT systems, (2) an overview of three initial approaches towards tackling some of the challenges in realising self-managing IoT systems, and (3) a set of open problems for future research on self-adaptation in the IoT domain.

The remainder of this paper is structured as follows. In Sect. 2, we provide a brief introduction to self-adaptation. Section 3 elaborates on the need for self-management in IoT and its specific challenges. Section 4 highlights a number of our initial efforts that aim to contribute towards tackling some of the challenges. Finally, Sect. 5 presents a set of open problems for future research in this area that we identified from our experiences.

2 Background on Self-adaptation

Dealing with uncertainties is an increasingly important challenge for software engineers. Here our focus is on the ability of software systems to deal with uncertainties that needs to be resolved at runtime [7,16,20]. A prominent approach to deal with uncertainties at runtime is so called self-adaptation [8,12,13,19,33]. Self-adaptation equips a software system with a feedback loop that collects data of the system and its environment that was difficult or impossible to determine before deployment. The feedback loop uses the collected data to reason about itself and to adapt itself to changes in order to provide the required quality goals, or gracefully degrade if needed. A typical example is a self-managing Web-based client-server system that continuously tracks and analyzes changes in work load and available bandwidth and dynamically adapts the server configuration to provide the required quality of service to its users, while minimising costs [8].

Self-adaptation can be considered from two perspectives [30]: (1) the ability of a system to adjust its behaviour in response to the perception of the environment and the system itself [3,15]; the *self* prefix indicates that the system decides and adapts autonomously (i.e., without or with minimal interference of humans) [2], and (2) the mechanisms that are used to realises self-adaptation, typically by means of a closed feedback loop [1,8,32], i.e. there is an explicit separation between a part of the system that deals with the domain concerns (goals for which the system is built) and a part that deals the adaptation concerns (the way the system realises its goals under changing conditions). Figure 1 shows the basic building blocks of a self-adaptive system, taken from [30].

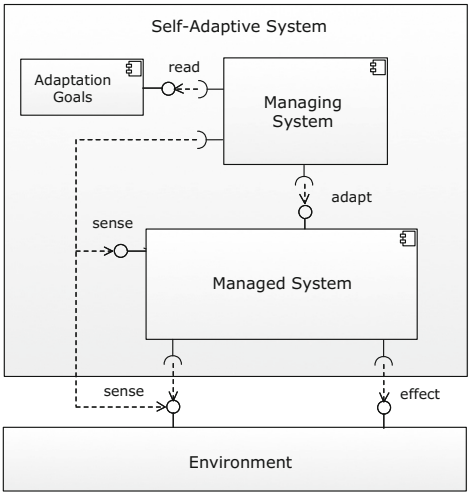

Fig. 1. Basic building blocks of a self-adaptive system [30]

The environment refers to the part of the external world with which the self-adaptive system interacts and in which the effects of the system can be observed [10]. The managed system comprises the application code that realizes the systems domain functionality. The managing system manages the managed system; that is, the managing system collects runtime data, reasons about this data and adapts the managed system to deal with one or more adaption goals. The adaptation goals are concerns of the managing system over the managed system; they usually relate to the software qualities of the managed system. Adaptation goals themselves can be subject of change (which is not shown in Fig. 1). A typical approach to structure the software of the managing system is by means of a so-called Monitor-Analyser-Planner-Executer + Knowledge feedback loop [4,12,26] (MAPE-K loop in short). The Monitor collects runtime data from the managed system and the environment and uses this to update the content of the Knowledge. Based on the current knowledge, the Analyser determines whether there is a need for adaptation of the managed system using the adaptation goals. If adaptation is required, the Planner puts together a plan that consists of a set of adaptation actions that are then enacted by the Executor that adapts the managed system as needed.

In the past few years, research in this area has particularly been focussing on how to provide assurances for the adaptation goals of self-adaptive systems that operate under uncertain operating conditions [14,31]. This is particularly important for systems with strict quality goals. Such systems require the provision of evidence that the system requirements are satisfied during its entire lifetime, from inception to and throughout operation. It is important to highlight that this evidence must be produced despite the uncertainty in the environment, the behaviour of the system itself and its requirements.

3 Why Do We Need Self-management in IoT?

IoT applications are inherently resource-constrained and subject to various types of dynamics during operation. These dynamics manifest themselves at different layers of the IoT technology stack. Figure 1 shows the typical layers of IoT applications. We highlight management concerns related to dynamics and uncertainties at different layers.

Things. The primary elements of IoT applications are battery powered motes. Consequently, energy consumption is a crucial aspect, as changing batteries is costly, or sometimes even not possible. The primary factor that determines energy consumption is communication, so the network should be configured carefully to avoid unnecessary communication. Motes of monitoring applications are equipped with sensors to sense the environment, such as RFID sensors, infrared and temperature sensors. However, IoT applications are not restricted to merely sensing and may also control elements in the environment, such as lightbulbs, heating devices, valves etc. Sensors and actuators are subject to all kind of uncertainties, ranging from inaccurate sampling or actuating up to failure.

Communication. IoT deployments primarily rely on wireless communication to relay sensor data to a central server. Wireless communication is subject to

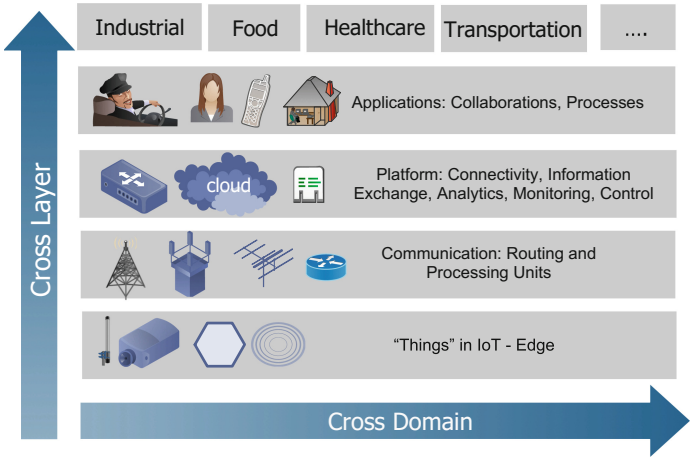

Fig. 2. Typical layers of IoT applications

runtime uncertainties, such as interferences, noise, and multi-path fading effects. Different communication technologies with dedicated protocols are applied to support different settings. For example, bluetooth enables establishing on the fly local networks between mobile entities, while a LoRa mesh network can support efficient very long range communication. Mobility may introduce particular challenges to reliable communication. Run-time uncertainties in communication result in packet loss. In such cases, it is important to reconfigure the wireless communication network to minimise packet loss (e.g. route messages differently in a multi-hop network setting).

Platform. Platforms provide the glue between user applications and the underlying IoT resources. An IoT platform offers a variety of services to applications and application developers, including a runtime environment, programming APIs etc. Platform services may range from the provision of basic resources to storage facilities up to advanced analytics and control management of underlying IoT resources. Crucial aspects in a distributed context are information exchange, monitoring and control services. Platforms can be deployed on various infrastructures, ranging from dedicated machines to a public cloud. IoT platforms and the infrastructure on which they are deployed can be subject of various sources of dynamics, typical examples are changes in the availability of resources, and dynamics in load (e.g. in a multi-tenant setting).

Applications. Application themselves can be subject of change, which in turn may affect the configuration of underlying layers. In the data-driven society, IoT deployments are acting as a catalyst to meet the demands of stakeholders in various disciplines. In the context of a smart city for example, garbage management units can support collection schedules for different parts of the city by knowing the status of individual garbage cans. Similarly, traffic regulators can dynamically alter the traffic routes in a city by knowing the traffic flows in

various parts of the city. These examples show that the sensor data produced by IoT deployments may be consumed by multiple stakeholders to tackle various societal issues. Such multidisciplinary approaches require the integration of domain specific knowledge into IoT deployments. Domain experts may modify their requirements during run-time to collect a particular type of sensor data with a specific setting.

Summary. We identified various concerns at different layers of the technology stack of IoT systems that require management. Often these concerns are handled either through over-provisioning (e.g. a conservative power settings of motes to ensure sufficient reliability), or through human intervention (e.g. an operator reconfigures the system to deal with temporal disruptions of service). Over-provisioning leads to inefficiencies and reduced lifetime of IoT systems. Manual intervention is not only very costly, it is also error prone. Hence, in order to fulfil the application demands and deal with continuous change and runtime uncertainty in a trustworthy manner, a self-management framework is essential.

4 Initial Contributions to Self-management in IoT

We highlight three initial contributions from our work that illustrate how self-adaptation techniques enable IoT systems to manage themselves autonomously. We start with Dawn that supports autonomous bandwidth allocation for IoT systems. Then we show how Hitch Hiker enables self-adaptation for concerns that cross multiple layers of IoT systems. Finally, we demonstrate how simulation and statistical techniques can be exploited at runtime to provide guarantees for a set of adaptation goals of an IoT application.

4.1 Autonomous Bandwidth Allocation Using Dawn

Dawn [24] is a self-management middleware for automatically configuring and reconfiguring 6TiSCH [28] networks based upon the requirements of their resident software. 6TiSCH [28] is a de-facto standard in high-reliability, low-power networking for the IoT. 6TiSCH networks are time synchronised and follow a communication schedule that repeats over time. The atomic unit of the communication schedule is a *time slot*. Time slots have a fixed, predefined duration, long enough for a single radio transmission and acknowledgment. Each platform is allocated a number of time slots in the schedule, that it then uses for communication. Platforms save energy by sleeping during inactive time slots. Each allocated time slot adds a quantum of communication bandwidth to the platform. The more time slots are allocated in the schedule for a given platform, the more data the platform can transmit per unit of time (higher bandwidth) and has more frequent transmission opportunities (lower latency), at the cost of higher energy consumption.

The schedule is typically created and maintained by an entity called the network manager. For periodic application traffic with static requirements, this process is straightforward. However, application dynamism, traffic periodicity

and traffic heterogeneity render approaches based on static bandwidth provisioning suboptimal. On one hand, over-provisioning bandwidth to account for the worst case increases energy consumption. On the other hand, under-provisioning bandwidth results in packet loss for non-deterministic traffic patterns due to insufficient bandwidth and thus lower reliability. The challenge is therefore to handle these non-deterministic traffic patterns while meeting requirements on low latency or high bandwidth, as well as the application dynamism that arises due to software and hardware reconfiguration. Practically, this means that each individual node should be provisioned with the optimum amount of bandwidth, and that this should be adjusted to meet the demands of runtime reconfiguration.

Dawn builds on top of LooCI binding model [9]. LooCI is a component based middleware for developing and managing IoT applications. In LooCI, application software is realized in the form of 'compositions' of reusable components. Figure 3 shows an example LooCI composition, where a temperature and light sensor component deployed on *Node A* communicate with an aggregator component on *Node B* via a TEMP and LIGHT type binding, respectively. A LooCI binding connects a component's provided interface (shown as –O) to another component's required interface (shown as)–), and it is depicted as –O)– in Fig. 3.

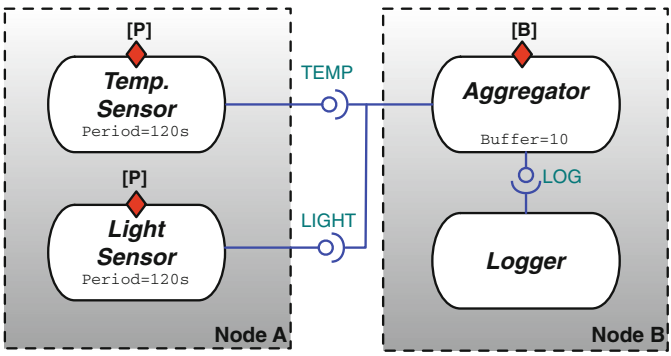

Fig. 3. Example composition of a component-based IoT application.

In the composition shown in Fig. 3, the temperature and light sensor components are sources, while the aggregator component is a sink. All *source* components are required to carry the standard Dawn property, *period* [P], which provides the transmission frequency of the source component in seconds. In this example, the temperature and light reading is transmitted once every 120 s. In cases where a *source* component transmits sporadically, such as the triggering of a PIR sensor, P provides the *maximum* rate at which the component may transmit. As with all LooCI properties, the *period* property may be inspected by external software or users. A component may also choose to allow runtime modification of this property in cases where transmission frequencies are determined by the application composition.

Intermediate components are located between a source and sink in the composition graph. These components may expose either the *period* property or the *buffer* property [B], which determines how many inputs the component will store from its dependent before forwarding a message. In the example composition shown in Fig. 3, the aggregator component buffers 10 sensor readings from the light and temperature interfaces before transmitting the aggregated results to the logger component. For each component in the composition with a *buffer* property [B], the bandwidth requirement of the component is the aggregate bandwidth requirement of its dependents. The property naming conventions such as *period* [P] and *buffer* property [B] are standardised in Dawn, and it enables Dawn to allocate optimal bandwidth for the compositions.

As can be seen from Fig. 3, the bandwidth requirement of a component depends on its *period* and *buffer* properties as well as the properties of all of its dependents in the component graph. The total bandwidth requirement of a node is therefore the aggregate of the transmission frequencies of all components with a *remote binding*. For the composition shown in Fig. 3, Node A has a bandwidth requirement of 8 bytes every 120 s, since the payload sizes of TEMP and LIGHT bindings are 5 and 3 bytes, respectively. However, Node B does not have any bandwidth requirement, as bandwidth assignments are based only on outgoing traffic and Node B has no remote bindings, thus there is no outgoing traffic. All the outgoing traffic on the node, which are determined by the remote bindings of components, require bandwidth resources from the network in order to reliably transmit the data to the intended destination. Existing bandwidth allocation approaches are static, which makes them suboptimal. In addition, such approaches offer less flexibility in the face of runtime reconfiguration.

Runtime reconfiguration of a software composition can significantly impact bandwidth requirements. Let us consider the software composition shown in Fig. 4, which is functionally equivalent to the composition shown in Fig. 3, but with a modified deployment location for the aggregator component and different *period* property settings.

While both compositions are functionally equivalent, the bandwidth allocation required to support the composition in Fig. 4 is 40 bytes every 50 min is 50 times less than the configuration shown in Fig. 3, since the buffer becomes full after receiving five sensor readings from both temperature and light sensor components. From these example compositions, it can be seen that the bandwidth requirement of the composition depends on the components and their properties. An automatic composition analysis approach is therefore required to extract bandwidth requirements from software compositions.

Dawn uses a composition analysis algorithm to derive the bandwidth requirements of application compositions, and then it invokes the bandwidth allocation algorithm to allocate the desired bandwidth for the IoT platform.

Dawn handles runtime reconfiguration by listening for reconfiguration actions at the middleware level. When reconfiguration is detected, the composition analysis and bandwidth allocation algorithms are executed. The process is fully automated and therefore imposes no burden on developers.

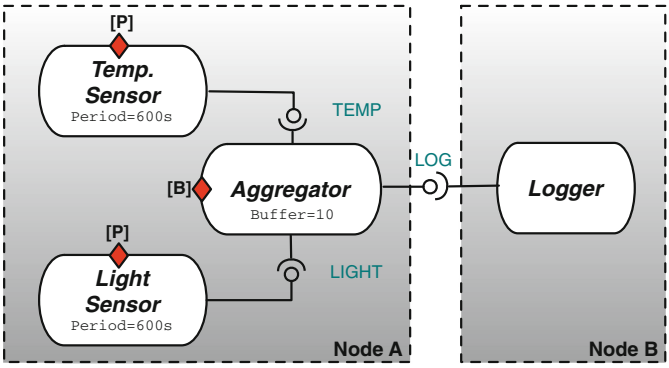

Fig. 4. Reconfigured composition of a component-based IoT application shown in Fig. 3.

Dawn elegantly automates the bandwidth reservation process, which enables the application developer to build extremely flexible and dependable IoT applications. Evaluation results on a 50-node testbed show that Dawn provides 100% reliability and manages to increase the lifetime by *three-fold* with minimal memory and performance overhead. For more information, we refer the interested reader to [24].

4.2 Self-adaptation Across Layers with Hitch Hiker

Internet-of-Things (IoT) devices must operate for long periods on limited power supplies. As discussed earlier, wireless communication is the primary source of energy consumption for IoT devices [22]. The lifetime of IoT applications can therefore be increased by minimising radio communication. Data aggregation has been widely applied to tackle this problem [11,23,27]. Data aggregation is a technique in which multiple messages are combined in to a single datagram, thus reducing radio transmissions and hence, the energy consumption of IoT devices. Furthermore, less frequent transmissions result in fewer collisions and therefore retransmissions. This can significantly improve the performance of IoT devices.

Hitch Hiker is a middleware that uses application knowledge to perform data aggregation based on the priority of the application data. Hitch Hiker allows the application developers to classify its application traffic as high-priority and low-priority based on its criticality. Hitch Hiker creates a data aggregation overlay using the high-priority transmissions, and the low-priority data is aggregated with high- priority transmissions. Hitch Hiker reduces the energy consumption, while offering a flexible data aggregation scheme for application developers.

Figure 5 shows the building blocks of Hitch Hiker distributed across the different layers of network stack. Hitch Hiker supports two types of management: centralized and decentralized. With the centralized scheme, the configuration and maintenance of low priority Hitch Hiker bindings is done by a centralized network manager. In this case, the central manager collects the information

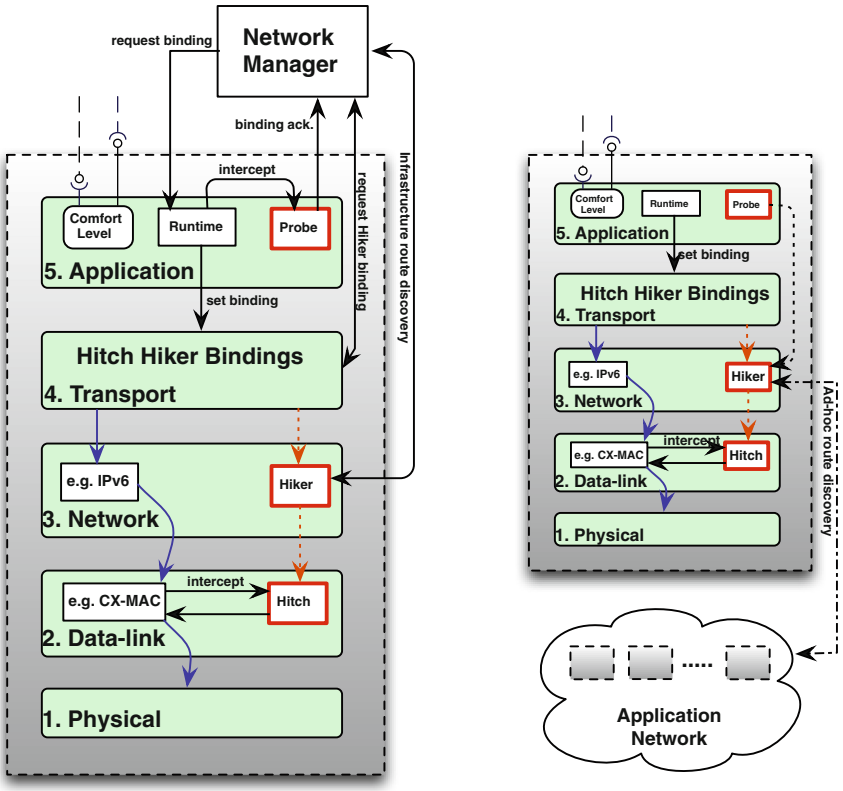

Fig. 5. High level overview of Hitch Hiker [25].

about high priority bindings, and it sets up the network for low priority data aggregation by configuring Hitch and Hiker protocols. In case of the decentralised scheme, the network configuration process is delegated to all the nodes in the network. Nodes self-configure themselves by coordinating and collaborating with each other.

Hitch Hiker autonomously add and remove data aggregation support for IoT applications using the existing high and low priority application bindings. Whenever the application gets reconfigured, Hitch Hiker recompute the route to retain the data aggregation functionality. If data aggregation route cannot be estabilished using the existing application compositions, Hitch Hiker notifies the application managers to take appropriate action.

Evaluation of the prototype implementation shows that Hitch Hiker consumes minimal memory, introduces limited overhead and that transmitting messages with Hitch Hiker consumes a small fraction of the energy that is required for a standard radio transmission. The interested reader finds more information about Hitch Hiker in [25].

4.3 Area Security Surveillance

In a joint R&D effort between imec-DistriNet and VersaSense[1] we studied self-adaptation for an area security surveillance application. The particular aim of this work was to evaluate whether simulation combined with statistical techniques can be used to provide guarantees for adaptation goals of an IoT system during operation. Figure 6 shows an overview of the deployment that a science campus of KU Leuven that we used in this study.

Fig. 6. Configuration area security surveillance application

The network is set up as a mesh network that comprises 15 motes equipped with different types of sensors that communicate over a time synchronised LoRa network. Motes are strategically placed to provide access control to labs (via RFID sensor), to monitor the movements and occupancy status (via Passive infrared sensor) and to sense the temperature (via heat sensor). The sensor data from all the motes are relayed to the IoT gateway, which is deployed at a central monitoring facility. The communication in the network is organised in cycles, each cycle comprising a fixed number of communication slots. Each slot defines a sender and receiver mote that can communicate with one another.

The domain concern for the IoT network is to relay surveillance data to the gateway. The stakeholders defined the adaptation goals as follows: (1) the average packet loss over 24 h should not exceed 10%, (2) the average latency of messages should be less than 5% of the cycle time, (3) the energy consumption of the motes should be minimised to optimise the life time of the network. Achieving these adaptation goals is challenging due to two primary types of

[1] www.versasense.com.

uncertainty: (1) network interference and noise caused by external factors such as weather conditions and the presence of other WiFi signals in the neighbourhood of communication links; interference affects the quality of the communication which may lead to packet loss; (2) fluctuating traffic load which may be difficult to predict (e.g., messages produced by a passive infrared sensor are based on the detection of motion of humans).

To solve the problem of the IoT network we applied a self-adaptation approach shown in Fig. 7.

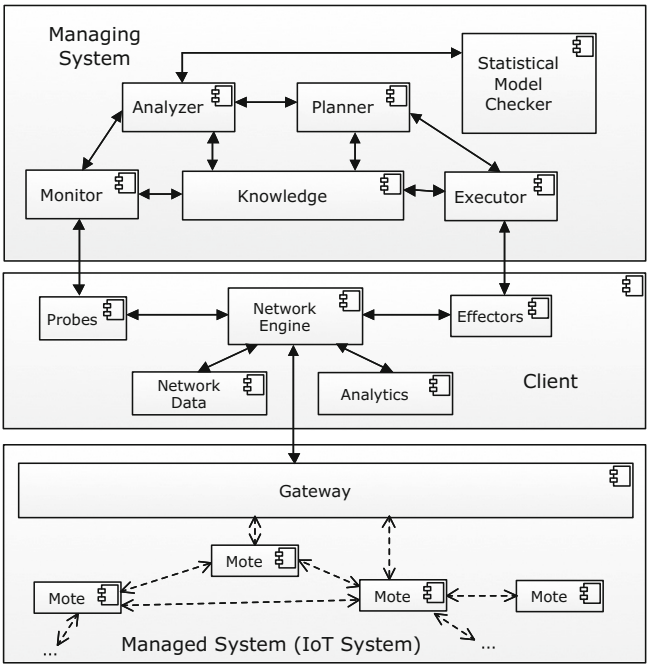

Fig. 7. Self-adaptation approach for the area security surveillance application

The bottom layer consists of the managed system with the network of motes and the gateway. The middle layer comprises a client that runs on a dedicated machine. This client offers an interface to the network using probes and effectors. Probes can be used to monitor the status of motes and links, statistical data about the packet loss, energy consumption, and latency of the network. The effectors allow adapting the mote settings, including power settings of the motes, distribution of messages to parents. The network engine collects the network data in a repository and performs analyses on the data to serve operators or adaptation logic using the analytics component. In manual mode, an operator can access the IoT network via the client to track its status and perform reconfigurations manually. These reconfigurations include changing the power settings

per communication link and changing the distribution of packets sent to parents (in case there are multiple parents). In the self-adaptive solution, the top layer is added to the system that automatically adapts the configuration such that the adaptation goals of the IoT network are met.

Self-adaptation is realised using a MAPE-K feedback loop. The Monitor uses the probe to track the recent traffic load and network interferences as well as the statistics for each quality property of interest. This data is used to update a set of models in the knowledge repository, including a model of the IoT system and its environment, a representation of the adaptation goals in the form of a set of rules, and a set of quality models, one for each adaptation goal.

The Analyzer uses a statistical model checker to predict the quality properties for each possible configuration of the IoT application. A configuration is characterised by: (i) a power setting for each communication link (a value between 0 and 15) and (ii) a distribution of packets sent along to links of motes with more than one parent (discretised in steps of 20%). The statistical model checker performs a series of simulations and uses statistical techniques to predict the qualities. Compared to exhaustive model checking, statistical model checking is very efficient in terms of verification time and required resources. The tradeoff is that the results are not exact, but subject to a level of confidence. The engineer can set this level, but higher confidence requires more time and resources. If the currently deployed configuration does not realise the adaptation goals, the planner is triggered to plan an adaptation. The results of analysis is a predicted value for each quality property of interest (average packet loss, average latency, energy consumption) for each possible configuration.

The Planner starts with selecting the best adaptation option based on the quality properties determined by the analyser. If valid configuration is found, a failsafe strategy is applied (i.e., the network is reconfigured to a default setting). Otherwise, the planner creates a plan to adapt the IoT network from its current configuration to the best adaptation option that was found. A plan consists of steps, where each step either adapts the power setting of a mote for a link, or it adapts the distribution of packets sent to a parent of a mote. As soon as the plan is ready, the Executer is triggered that will enact the adaptation steps via the effectors.

We compared the self-adaptation approach with an approach commonly used in practice that uses over-provisioning to deal with uncertainties (power settings are set to maximum and packets are duplicated in case of multiple parents).

We evaluated the packet loss, latency, and energy consumption of the IoT network for both approaches for a period of 24 h. The cycle time was set to 9.5 min, corresponding to 153 cycles in 24 h. During the first 8 min of the cycle the motes can communicate packets downstream to the gateway; during the remaining 1.5 min the gateway can communicate adaptation messages upstream to the motes. For the self-adaptation approach we configured the verification queries with a confidence of 90% and simulations queries with a relative standard error of the mean of 0.5%. Figure 8 shows the main results.

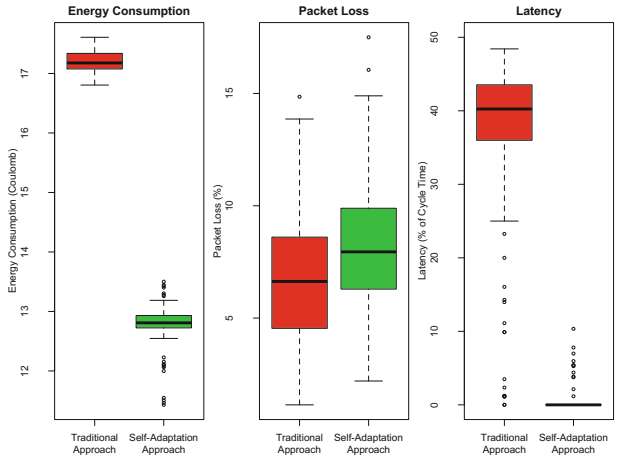

Fig. 8. Test results for the area security surveillance application

The graphs show that the average energy consumption of the self-adaptation solution is significantly better compared to the traditional approach (p-value < 0.000). Similarly, the self-adaptation approach outperforms the traditional approach for latency (p-value < 0.000). For the packet loss, both approaches have similar results (mean of paired differences is 1.4%). We measured also the time required for adaptation. With a mean of 45.7 s, the adaptation time was perfectly fine for a setting with a cycle time of around 9.5 min with 1.5 min to make an adaptation decision.

The area security surveillance application demonstrates how self-adaption techniques can be applied to enable an IoT application to deal with uncertainties at runtime and provide guarantees with sufficient confidence for a set of required quality properties in an automatic manner. For more information, we refer the reader to the DeltaIoT website[2].

5 Open Problems for Self-management in IoT

We conclude this paper with a number of open challenges for future research on self-adaptation of IoT systems that we identified based on the state of the art and our experiences with engineering concrete IoT applications.

Local adaptation. The examples of autonomous bandwidth allocation using Dawn (Sect. 4.1) and area security surveillance (Sect. 4.3) are examples of self-adaptation that is applied locally. Existing solutions such as these have primarily focussed on the benefits and tradeoffs in terms of qualities that can be achieved by self-adaptation. However, in the context of IoT, an important factor is the cost associated with applying the adaptation actions. For example, in a mesh

[2] https://people.cs.kuleuven.be/danny.weyns/software/DeltaIoT/.

network, to adjust the network settings of motes, adaptation messages needs to be routed from the gateway upstream to the motes. Communicating these messages requires energy. The cost of this energy may invalidate the expected benefits of the adaptation. Another example of cost may be the time that is required to enact the adaptation actions. Hence, an important challenge for future research is to develop solutions that consider both the benefits and the costs of self-adaptation.

Cross-layer adaption. Hitch Hiker is an approach that supports cross-layer adaptation (Sect. 4.2). In the context of smart cities, IoT applications typically consist of hundreds of motes equipped with various types of sensors and actuators. Continuous adaptation based on context changes of such motes has shown to be useful for understanding sensor data [23]. On the other hand, reconfigurations of applications may also alter the underlying communication demands [24,25]. As a consequence, reconfiguration at one layer of the technology stack may call for reconfigurations at another layer. While traditional layering schemas leads to separation of concerns, it may be less suitable for dynamic IoT applications where concerns inevitably crosscut the layers. An important challenge is to investigate how to deal with dominant crosscutting concerns such as energy efficiency and security in IoT, which may require a new view on layering of the IoT technology stack.

Cross-application adaption. Besides adaptation concerns that span different layers of the IoT technology stack, concerns can also cross domains as shown in Fig. 2. Although generally considered as crucial for the future of IoT, little research has been devoted to interactions and collaborations between different IoT applications. Such collaborations have the potential to generate dramatic synergies [6]. However, at the same time they create dependencies that in a dynamic context may be extremely difficult to handle. Hence, an important challenge for future research is how to investigate the interplay between IoT applications in an ecosystem. This will require solutions for technical alignment and stability, but also suitable business models and methods for establishing trust.

Providing guarantees. One of the crucial aspects of many IoT applications is trustworthiness. Trustworthiness refers to stakeholders' confidence, dependability, and reliability in the applications. As we have highlighted in Sect. 3, given that IoT applications are subject to a zoo of uncertainties, this raises an important challenge: how to obtain trustworthiness in IoT systems that are subject of ongoing uncertainties? Tacking this challenge is hard, in particular in an ecosystem context. It does not only require novel technical solutions to guarantee the concerns of stakeholders throughout the lifetime of IoT systems, it also requires novel legal frameworks that can handle continuous change.

Acknowledgments. We are grateful to the technical staff of VersaSense (https://www.versasense.com/) for the fruitful collaborations.

References

1. Andersson, J., de Lemos, R., Malek, S., Weyns, D.: Modeling dimensions of self-adaptive software systems. In: Cheng, B.H.C., de Lemos, R., Giese, H., Inverardi, P., Magee, J. (eds.) Software Engineering for Self-adaptive Systems. LNCS, vol. 5525, pp. 27–47. Springer, Heidelberg (2009). https://doi.org/10.1007/978-3-642-02161-9_2
2. Brun, Y., et al.: Engineering self-adaptive systems through feedback loops. In: Cheng, B.H.C., de Lemos, R., Giese, H., Inverardi, P., Magee, J. (eds.) Software Engineering for Self-adaptive Systems. LNCS, vol. 5525, pp. 48–70. Springer, Heidelberg (2009). https://doi.org/10.1007/978-3-642-02161-9_3
3. Cheng, B.H.C., et al.: Software engineering for self-adaptive systems: a research roadmap. In: Cheng, B.H.C., de Lemos, R., Giese, H., Inverardi, P., Magee, J. (eds.) Software Engineering for Self-adaptive Systems. LNCS, vol. 5525, pp. 1–26. Springer, Heidelberg (2009). https://doi.org/10.1007/978-3-642-02161-9_1
4. Dobson, S., Denazis, S., Fernández, A., Gaïti, D., Gelenbe, E., Massacci, F., Nixon, P., Saffre, F., Schmidt, N., Zambonelli, F.: A survey of autonomic communications. ACM Trans. Auton. Adapt. Syst. **1**(2), 223–259 (2006). http://doi.acm.org/10.1145/1186778.1186782
5. Dohler, M., Barthel, D., Watteyne, T., Winter, T.: RFC5548: routing requirements for urban low-power and lossy networks (2009)
6. Dustdar, S., Nastic, S., Scekic, O.: A novel vision of cyber-human smart city. In: 2016 Fourth IEEE Workshop on Hot Topics in Web Systems and Technologies (HotWeb), pp. 42–47, October 2016
7. Esfahani, N., Malek, S.: Uncertainty in self-adaptive software systems. In: de Lemos, R., Giese, H., Müller, H.A., Shaw, M. (eds.) Software Engineering for Self-adaptive Systems II. LNCS, vol. 7475, pp. 214–238. Springer, Heidelberg (2013). https://doi.org/10.1007/978-3-642-35813-5_9
8. Garlan, D., Cheng, S., Huang, A., Schmerl, B., Steenkiste, P.: Rainbow: architecture-based self-adaptation with reusable infrastructure. Computer **37**(10), 46–54 (2004)
9. Hughes, D., Thoelen, K., Maerien, J., Matthys, N., Del Cid, J., Horre, W., Huygens, C., Michiels, S., Joosen, W.: LooCI: the loosely-coupled component infrastructure. In: Proceeding of the 11th IEEE International Symposium on Network Computing and Applications, pp. 236–243 (2012)
10. Jackson, M.: The meaning of requirements. Ann. Softw. Eng. **3**, 5–21 (1997). http://dl.acm.org/citation.cfm?id=590564.590577
11. Kalpakis, K., Dasgupta, K., Namjoshi, P.: Maximum lifetime data gathering and aggregation in wireless sensor networks. Proc. IEEE Netw. **2**, 685–696 (2002)
12. Kephart, J., Chess, D.: The vision of autonomic computing. Computer **36**(1), 41–50 (2003)
13. Kramer, J., Magee, J.: Self-managed systems: an architectural challenge. In: Future of Software Engineering, FOSE 2007. IEEE Computer Society (2007)
14. de Lemos, R., et al.: Software engineering for self-adaptive systems: research challenges in the provision of assurances. In: de Lemos, R., Garlan, D., Ghezzi, C., Giese, H. (eds.) Software Engineering for Self-adaptive Systems III. LNCS, vol. 9640. Springer, Heidelberg (2018, forthcoming). https://people.cs.kuleuven.be/danny.weyns/papers/2018SEfSAS.pdf

15. de Lemos, R., et al.: Software engineering for self-adaptive systems: a second research roadmap. In: de Lemos, R., Giese, H., Müller, H.A., Shaw, M. (eds.) Software Engineering for Self-adaptive Systems II. LNCS, vol. 7475, pp. 1–32. Springer, Heidelberg (2013). https://doi.org/10.1007/978-3-642-35813-5_1
16. Mahdavi-Hezavehi, S., Avgeriou, P., Weyns, D.: A classification of current architecture-based approaches tackling uncertainty in self-adaptive systems with multiple requirements. In: Managing Trade-offs in Adaptable Software Architectures. Elsevier (2016)
17. Mainwaring, A., Culler, D., Polastre, J., Szewczyk, R., Anderson, J.: Wireless sensor networks for habitat monitoring. In: Proceedings of the 1st ACM International Workshop on Wireless Sensor Networks and Applications, WSNA 2002, pp. 88–97. ACM, New York (2002). http://doi.acm.org/10.1145/570738.570751
18. Martocci, J., Mil, P., Riou, N., Vermeylen, W.: Building automation routing requirements in low-power and lossy networks (5867) (2010)
19. Oreizy, P., Medvidovic, N., Taylor, R.: Architecture-based runtime software evolution. In: International Conference on Software Engineering, ICSE 1998. IEEE Computer Society (1998). http://dl.acm.org/citation.cfm?id=302163.302181
20. Perez-Palacin, D., Mirandola, R.: Uncertainties in the modelling of self-adaptive systems: a taxonomy and an example of availability evaluation. In: International Conference on Performance Engineering, ICPE 2014 (2014)
21. Pister, K., Thubert, P., Dwars, S., Phinney, T.: Industrial routing requirements in low-power and lossy networks. Technical report (2009)
22. Raghunathan, V., Schurgers, C., Park, S., Srivastava, M.: Energy-aware wireless microsensor networks. IEEE Sig. Process. Mag. **19**(2), 40–50 (2002)
23. Rajagopalan, R., Varshney, P.: Data-aggregation techniques in sensor networks: a survey. IEEE Commun. Surv. Tutor. **8**(4), 48–63 (2006)
24. Ramachandran, G.S., Matthys, N., Daniels, W., Joosen, W., Hughes, D.: Building dynamic and dependable component-based internet-of-things applications with dawn. In: 2016 19th International ACM SIGSOFT Symposium on Component-Based Software Engineering (CBSE), pp. 97–106, April 2016
25. Ramachandran, G.S., Proenca, J., Daniels, W., Pickavet, M., Staessens, D., Huygens, C., Joosen, W., Hughes, D.: Hitch hiker 2.0: a binding model with flexible data aggregation for the internet-of-things. J. Internet Serv. Appl. **7**(1), 4 (2016). http://dx.doi.org/10.1186/s13174-016-0047-7
26. Salehie, M., Tahvildari, L.: Self-adaptive software: landscape and research challenges. Trans. Auton. Adapt. Syst. **4**, 14:1–14:42 (2009)
27. Tan, H.O., Körpeoğlu, I.: Power efficient data gathering and aggregation in wireless sensor networks. SIGMOD Rec. **32**(4), 66–71 (2003). http://doi.acm.org/10.1145/959060.959072
28. Watteyne, T., Palattella, M., Grieco, L.: Using IEEE 802.15.4e time-slotted channel hopping (TSCH) in the Internet of Things (IoT): problem statement. RFC 7554, RFC Editor, May 2015
29. Watteyne, T., Weiss, J., Doherty, L., Simon, J.: Industrial IEEE802.15.4e networks: performance and trade-offs. In: 2015 IEEE International Conference on Communications (ICC), pp. 604–609, June 2015
30. Weyns, D.: Software engineering of self-adaptive systems: an organised tour and future challenges. In: Dick Taylor, R., Kang, K., Cha, S. (eds.) Handbook of Software Engineering. Springer, Heidelberg (2018, forthcoming). https://people.cs.kuleuven.be/danny.weyns/papers/2017HSE.pdf

31. Weyns, D., et al.: Perpetual assurances in self-adaptive systems. In: de Lemos, R., Garlan, D., Ghezzi, C., Giese, H. (eds.) Software Engineering for Self-adaptive Systems III. LNCS, vol. 9640. Springer, Heidelberg (2018, forthcoming). https://people.cs.kuleuven.be/danny.weyns/papers/2016SEfSAS.pdf
32. Weyns, D., Iftikhar, U., Söderlund, J.: Do external feedback loops improve the design of self-adaptive systems? A controlled experiment. In: International Symposium on Software Engineering of Self-managing and Adaptive Systems, SEAMS 2013 (2013)
33. Weyns, D., Malek, S., Andersson, J.: FORMS: unifying reference model for formal specification of distributed self-adaptive systems. ACM Trans. Auton. Adapt. Syst. **7**(1), 8:1–8:61 (2012)