Designing and Implementing Resilient IoT Applications in the Fog: A Smart Home Use Case

Umar Ozeer, Loïc Letondeur, François-Gaël Ottogalli, Gwen Salaün, Jean-Marc Vincent

Designing and Implementing Resilient IoT Applications in the Fog: A Smart Home Use Case

Umar Ozeer, Loïc Letondeur, François-Gaël Ottogalli
Orange Labs
Meylan, France
firstname.lastname@orange.com

Gwen Salaün, Jean-Marc Vincent
Univ. Grenoble Alpes, CNRS, Inria, LIG
Grenoble, France
firstname.lastname@inria.fr

***Abstract*—Fog computing extends the capacities of the cloud to the edge of the network, near the physical world, so that Internet of Things (IoT) applications can benefit from properties such as short delays, real-time and privacy. Devices in the Fog-IoT environment are usually unstable and prone to failures. In this context, the consequences of failures may impact the physical world and can, therefore, be critical. This paper reports a framework for end-to-end resilience of Fog-IoT applications. The framework was implemented and experimented on a smart home testbed.**

***Index Terms*—Smart Home; Resilience; Fog Computing; IoT**

I. Introduction

Fog computing [1] meets the requirements of IoT applications such as low latencies, real-time, privacy, data analysis and filter at the edge, near the physical world (PW), which the cloud fails to provide. Since the Fog is highly heterogeneous, dynamic and involves cyber-physical interactions, it brings new challenges regarding the design of resilient applications. Devices and network channels in the Fog are usually unstable and prone to failures. This is a result of bulk production and cheap design. In addition, devices usually suffer from external PW environmental conditions which increase probability of failures. Failures in this context may have consequences on the PW which can be potentially critical. For instance, the failure of a smoke detector in a building can be hazardous. Furthermore, there is a need for state restoration when recovering from failures to keep consistency with the PW. For example, the state of a drug injection device for patients should be restored after a failure to prevent injection of already administered drugs.

This paper describes our resilience framework that enables IoT applications developers to provide resilient services in a dynamic, heterogeneous and cyber-physical Fog-IoT environment. The framework limits the propagation of failures, avoids the restart of the whole application when a failure occurs, and, recovers from both infrastructure and applicative entities by reconfiguring and restoring a consistent state of the application, including consistency with respect to the PW (PW-consistency). The framework was implemented and experimented on a testbed, inspired from [2], reproducing a real life smart home application. Section II showcases the smart home testbed and Section III details the resilience framework and how it is implemented on this use case application.

II. Experimental Environment

The target testbed is a smart home application for light automation and physical intrusion detection.

A. Infrastructure and Applicative Entities of the Testbed

The infrastructure of the testbed is composed of a set of *Physical Nodes* and *Appliances*. The Physical Nodes PC1, rpi01, rpi2 and rpi3 are, respectively, a PC (x86_64, 4GB RAM, Windows 7), a Raspberry Pi Model Zero (32-bit, 1GHz, single-core ARM11 processor, 512 MB RAM, 16GB microSD storage, Raspbian GNU/Linux 8.0 jessie) and two Raspberry Pi Model 3 Type B (32-bit, 1.2 GHz, quad-core ARM Cortex-A53 processor, 1 GB RAM, 16GB microSD storage, Raspbian GNU/Linux 8.0 jessie). These devices are representative of the capacities of typical smart home devices and are readily available at cheap prices. Each Physical Node hosts an administrative entity, namely a *Fog Node* which gives access to the underlying resources of the Physical Node and provides the runtime for the execution of software entities. Each Fog Node hosts a *Fog Agent* for handling the lifecycle operations on software entities. A house with three rooms is considered in this demo. Figure 1 shows the distribution of the devices in the house as well as the placement of the software entities hosted on the Fog Nodes. Appliances are: two Philips Hue Lamps, a Hue Go Lamp (bedside lamp), a Hue

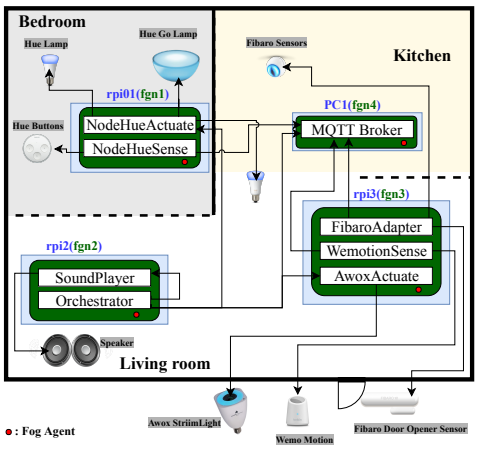

Fig. 1. Smart Home Infrastructure and Application

Tap (a set of four connected buttons), a Fibaro Multipurpose Sensor (motion, light, temperature and vibration sensors), an analog wired Speaker, an Awox Striimlight Lamp (lamp with integrated speaker), a Wemo Motion Sensor and a Fibaro Door Opener Sensor. The Philips Hue devices are connected to a bridge through the wireless protocol Zigbee. The Fibaro devices uses Z-wave protocol. Awox Striimlight and Wemo Motion are connected through Wi-Fi.

The software entities are:

- *MQTT Broker*: a Message Oriented Middleware (MOM) based on a publish-subscribe communication pattern. It is an implementation of a MQTT broker based on ActiveMQ [3].
- *Orchestrator*: it subscribes to all the events published into the MQTT Broker. It defines the corresponding scenarios (set of actions) that should be triggered based on patterns of events that are reported by sensors. It sends messages to other software entities according to the scenarios it implements. It is developed in *Node.js*.
- *NodeHueSense*: it reports events from the Hue buttons and publishes them on the MQTT bus. The event is the corresponding button pressed. It is developed in *Node.js*.
- *FibaroAdapter*: it reports the events sensed by Fibaro devices. It can also configure the frequency of reported events from the devices. The events sensed are published on the MQTT bus.
- *WemotionSense*: it reports motion events sensed by the Wemo Motion device and publishes them onto the MQTT bus. It is developed in Go.
- *NodeHueActuate*: it accepts messages relative to the control of the Hue lamps. The latter are controlled via the REST API they expose. It is written in *Node.js*.
- *AwoxActuate*: it accepts messages for the control of the Awox Striimlight lamp and its integrated speaker. The lamp is controlled via its SOAP API. It is developed in Go.
- *SoundPlayer*: it accepts messages for the actuation of the Speaker. It is based on the open-source audio player *mpg123* [4].

Node.js was preferred for development as it is lightweight, allows asynchronous operations and its packet manager, *npm*, handles effectively the management of runtime dependencies which makes its integration simple in this environment.

B. User Stories

Two main user stories are implemented. In the *Bedtime Scenario*, a button press in the bedroom indicates the bedtime of the house tenant. All the lights of the house are turned off and an alarm is set. If the door is opened or motion is detected in the living room or the kitchen, the alarm is triggered on the speaker and the lamps of the house are turned on in a red colour. In the *Welcome Home Scenario*, the Wemo Motion reports motion when the home tenant arrives at the front door. The lamp at the entrance is turned on, allowing the person to unlock the door. Upon entering the house, the living room lamp is turned on and a greeting sound is played on the speaker. More scenarios can be composed by the Orchestrator based on the patterns of events sensed and actuated.

III. Resilience Approach and Framework

The resilience approach consists of four functional steps: (i) State Saving, (ii) Monitoring, (iii) Failure Notification and Reconfiguration, and, (iv) Recovery. [5] describes in details the resilience approach. In a first step, the states of applicative entities and the PW are saved in an uncoordinated way. The second step involves the monitoring of both infrastructure and applicative entities for failure and recovery detection. When a failure is detected, failure notifications are propagated to reconfigure the application with respect to that failure. In the last step, the data saved in step one are used to restore a consistent and stable state of the application, including PW-consistency.

A. Management Entities and Framework APIs

In order to ensure these steps and functionalities, a resilience framework was developed, providing (i) a set of management entities as well as (ii) APIs which allow developers to extend the functions of the software entities and Fog Agents for resilience purposes. Figure 2 shows the deployment of the management entities on the use case infrastructure:

1) a *Global Manager, GM,* which takes global decisions relative to failure detection and rules for recovery,
2) a stable storage based on *MongoDB* for persisting states data,
3) the Applicative Lifecycle Manager [2], ALM, for the lifecycle management of software entities, and,
4) *Thing'in* [6], a digital index of connected devices and their relationships.

These management entities are deployed on a *RuggedPOD* [7], as illustrated in Figure 2, which is considered to be reliable. The RuggedPOD is a micro-datacenter and is water-proof, passively cooled server with four 8 cores *Xeon* CPUs that can be placed in the neighbourhood as part of a Telco's infrastructure. In the case of software entities, besides their business APIs, two classes of APIs are provided by the framework:

- *SaveStateAPI* for the definition of the state data and the techniques of saving. The aim is to save data (e.g.: tuning parameters, environment variables, dependencies, configuration files, function calls and messages) using techniques of uncoordinated checkpoint [8], [9], message logs [10], [11] and function-call records to save the current state of applicative entities. The state of the PW is given by the events sensed and actuated. These data are saved on the stable storage.
- *ConfigAPI* for processing failure notifications and for reconfiguring the software entities with respect to failures. Reconfiguration aims at limiting the propagation of failures by adapting the functional behaviour of the software entities. For instance, reconfiguration may involve a temporary pause in execution or the stop of processing events to and from an entity suspected of failure.

In the case of Fog Agents, beside lifecycle operations instructed by the ALM, two additional classes of APIs for

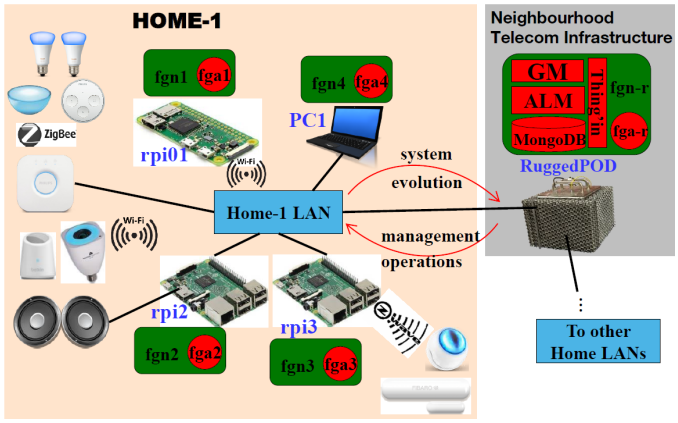

Fig. 2. Deployment of Management Entities on the Smart Home Infrastructure

failure management are provided: *(i) MonitorAPI* implements four types of monitoring: local system observation, heartbeat, ping-ack, and applicative/control message observation. The techniques are chosen to limit influence on the execution of the application and on the network. For example, local system observation is used to monitor software entities and applicative/control message observation for appliances that communicate regularly (Fibaro Multipurpose Sensor) to avoid unnecessary messages exchanges on the network. In other cases, monitoring by ping-ack is unavoidable (Hue Lamps, Awox Striimlight and WeMo Motion). Each Fog Agent also implements a heartbeat to its neighbouring Fog Agents for the monitoring of the Physical Nodes. A Fog Agent reports to GM any entity suspected of failure and their subsequent recovery. *(ii) StateMgtAPI* for handling the messages received from GM for state restoration during recovery. As illustrated in Figure 2, the evolution of the infrastructure and application in terms of failure, recovery, state change are reported to the management entities which in turn performs administrative operations.

B. Failure Scenarios and Recovery

This Section aims at describing how recovery is performed for different types of failures.

Appliance Failure and PW-Consistent Recovery. The failure of the main lamp (Hue Lamp) in the bedroom corresponds to the burn out of the lamp or to an electrical power cut.

- *fga1* detects the failure of the lamp and sends a failure notification message to GM.
- GM queries Thing'in for a functionally equivalent Appliance, which in this case returns the bedside lamp.
- Failure notifications are propagated so that the bedside lamp ensures the functions of the main lamp. This means that the buttons used to turn on and off the main lamp will now do so for the bedside lamp.
- The state of the failed lamp is retrieved by GM from the stable storage and sent to *fga1*.
- *fga1* restores the state of the failed lamp onto the bedside lamp. The state pushed can turn on/off the lamp or set it to a specific colour. A recovery message is then sent to GM.

Software Entity Failure and Recovery on the Same Fog Node. The failure of *SoundPlayer* is targeted.

- *fga2* detects the failure of *SoundPlayer* and sends a failure message to GM.
- Failure notifications are propagated to software entities.
- GM sends a message to *fga2* to restart *SoundPlayer* on the same Fog Node, *fga2*, in a pause mode.
- GM retrieves the state of *SoundPlayer* from the stable storage and sends it to *fga2*.
- *fga2* restores the state of *SoundPlayer*, resumes its execution, and sends a recovery message to GM.
- Recovery notifications are propagated to software entities.

Physical Node Failure and Redeployment of Software Entities. The failure of the Physical Node *rpi01* is targeted.

- *fga2*, *fga3* and *fga4* detect the failure of *rpi01* and send failure messages to GM.
- Failure notifications are propagated to software entities.
- GM requests a new placement and deployment for *NodeHueActuate* and *NodeHueSense* to the ALM, which returns for instance *fgn2*.
- Both software entities are redeployed on *fgn2*, their state restored (by *fga2*) and recovery notifications propagated as in the previous case.

IV. Conclusion

This paper reports the design and implementation of a resilience approach on a realistic testbed in a Fog-IoT smart home environment. The resilience approach is designed taking into account the specificities of the environment. The practical experiments show the feasibility of the approach and that recovery is achieved in an acceptable delay from a user point of view [5]. Future work includes a performance evaluation on different testbeds and formal verification of the resilience protocol.

References

[1] F. Bonomi, R. Milito, J. Zhu, and S. Addepalli, "Fog computing and its role in the internet of things," in *Proc. of MCC'12*. ACM, 2012, pp. 13–16.
[2] L. Letondeur, F.-G. Ottogalli, and T. Coupaye, "A demo of application lifecycle management for iot collaborative neighborhood in the fog," in *IEEE Fog World Congress*. IEEE, 2017, pp. 1–6.
[3] ActiveMQ, https://activemq.apache.org, online; accessed 30/07/2018.
[4] mpg123 Audio Player, www.mpg123.de, online; accessed 30/07/2018.
[5] U. Ozeer, X. Etchevers, L. Letondeur, F.-G. Ottogalli, G. Salaün, and J.-M. Vincent, "Resilience of stateful IoT applications in a dynamic fog environment," in *Proc. of 15th EAI Mobile and Ubiquitous Systems: Computing, Networking and Services*. ACM, 2018, pp. 332–341.
[6] Thing'in, http://thinginthefuture.com, online; accessed 20/09/2018.
[7] RuggedPOD, http://ruggedpod.qyshare.com, online; accessed 30/07/2018.
[8] E. N. M. Elnozahy, L. Alvisi, Y.-M. Wang, and D. B. Johnson, "A survey of rollback-recovery protocols in message-passing systems," *ACM Comput. Surv.*, vol. 34, no. 3, pp. 375–408, 2002.
[9] A. Khunteta and K. Praveen, "An analysis of checkpointing algorithms for distributed mobile systems," *International Journal on Computer Science and Engineering*, vol. 2, 2010.
[10] L. Alvisi and K. Marzullo, "Message logging: pessimistic, optimistic, causal, and optimal," *IEEE Trans. on Software Engineering*, vol. 24, no. 2, pp. 149–159, 1998.
[11] R. Strom and S. Yemini, "Optimistic recovery in distributed systems," *ACM Trans. Comput. Syst.*, vol. 3, no. 3, pp. 204–226, 1985.