Towards a Resilient Smart Home

Tam Thanh Doan, Reihaneh Safavi-Naini, Shuai Li, Sepideh Avizheh,
Muni Venkateswarlu K., Philip W. L. Fong
University of Calgary, Canada

ABSTRACT

Today's Smart Home platforms such as Samsung SmartThings and Amazon AWS IoT are primarily cloud based: devices in the home sense the environment and send the collected data, directly or through a hub, to the cloud. Cloud runs various applications and analytics on the collected data, and generates commands according to the users' specifications that are sent to the actuators to control the environment. The role of the hub in this setup is effectively message passing between the devices and the cloud, while the required analytics, computation, and control are all performed by the cloud. We ask the following question: what if the cloud is not available? This can happen not only by accident or natural causes, but also due to targeted attacks. We discuss possible effects of such unavailability on the functionalities that are commonly available in smart homes, including security and safety related services as well as support for health and well-being of home users, and propose RES-Hub, a hub that can provide the required functionalities when the cloud is unavailable. During the normal functioning of the system, RES-Hub will receive regular status updates from cloud, and will use this information to continue to provide the user specified services when it detects the cloud is down. We describe an IoTivity-based software architecture that is used to implement RES-Hub in a flexible and expendable way and discuss our implementation.

KEYWORDS

IoT Security, Smart home resiliency, Smart home security

ACM Reference Format:
Tam Thanh Doan, Reihaneh Safavi-Naini, Shuai Li, Sepideh Avizheh, Muni Venkateswarlu K., Philip W. L. Fong . 2018. Towards a Resilient Smart Home . In *IoT S&P'18: ACM SIGCOMM 2018 Workshop on IoT Security and Privacy , August 20, 2018, Budapest, Hungary*. ACM, New York, NY, USA, 7 pages. https://doi.org/10.1145/3229565.3229570

1 INTRODUCTION

A smart home is equipped with hundreds of sensors that perform measurements, and in combination with other data sources (e.g. third party data), provides smarts that are used for personalization and automated services, as well as improving efficiency and convenience of its residents. Sensors carried by home users (residents and visitors) can also be integrated into the home network to enhance their service offerings. A smart home can be seen as a full-fledged integrated computing system that senses the environment (inside and outside the home), performs computation, and generates the commands that are used to control the environment, or provides services that are requested by the users [13].

Cloud-based Smart Homes. Today's smart home platforms are primarily cloud-based: that is the required computation, analytics and generation of smarts are performed at the cloud provider and appropriate results are sent back. Established smart home solution providers, including Samsung SmartThings [23], Amazon AWS (Amazon Web Services) [8], IBM Watson [18] and Microsoft Azure [20], rely on cloud services. The home sensed data are either directly sent to the cloud (e.g. Amazon AWS), or sent through an intermediate *hub* whose primary role is to provides connectivity to cloud for the home sensors (e.g. Samsung SmartThings). Processing data in cloud allows incorporating data from other sources (e.g. homes in the same area, or other clouds) be used for powerful analytics and visualization applications that are available to cloud.

Reliance on cloud however raises many security and privacy, as well as reliability challenges. Existing smart home platforms all assume full trust on the cloud provider which has full access (and control) to fine-grained private information of the home users. Data on user movements, requested services from devices inside the home, or providers outside, as well as users' environmental preferences can be easily gleaned from the collected data. Although in some frameworks such as IBM Watson IoT, one can encrypt the sensed data before sending them to cloud, this will remove the benefits of using many of the cloud services. This was the motivation for a host-based architecture [13] where data storage and processing are performed within the home, with cloud used as backup.

In this paper we consider the reliability aspect of full reliance on cloud (and Internet connectivity): we ask the question, "what happens to a smart home and its users, when it does not have access to cloud?" This can be the result of the network being down or cloud services being unavailable, and will become particularly important when the home is vacant and the disconnection remains un-noticed for an extended period of time. Disconnection can happen because of natural causes (e.g. storm or flood) or a malicious attacker disconnecting the home for malicious reasons (e.g. burglary, or providing a time window to tamper with the house for future attacks).

Our Contributions. We discuss possible effects of cloud unavailability on the functionalities that are commonly available in smart homes, including security and safety related services as well as support for health and well-being of home users. We propose RES-Hub (Resilient Home-Hub), a hub that can provide the designed functionalities when the cloud is unavailable. RES-Hub will receive regular status updates from cloud during the normal functioning of the system, and will use this information to continue to provide the user specified (essential) services when it detects cloud is unavailable. We describe an IoTivity-based software architecture that

IoT S&P'18, August 20, 2018, Budapest, Hungary

ACM ISBN 978-1-4503-5905-4/18/08…$15.00
https://doi.org/10.1145/3229565.3229570

can be used to implement RES-Hub in a secure and extendable way, with flexible access. IoTivity [19] is an open source project that is supported by a large group of manufacturers. By "secure" we mean RES-Hub will ensure correct functioning and controlled access to the home resources, albeit with a user specified subset of functionalities, when the cloud is unavailable. By extendability we mean the RES-Hub can be used with devices of different manufacturers and will not depend on a particular manufacturer. Flexible control means that the hub can be remotely accessed, allowing the user to control the home from outside. We discuss our proof of concept implementation of RES-Hub and its secure implementation using OAuth 2.0 for authentication and authorization.

2 SMART HOME & CLOUD DISCONNECTION

We first briefly look at important existing smart home platforms and the effect of Internet disconnection on their operation.

Samsung SmartThings [22] platform is for developing IoT applications for smart homes. SmartThings ecosystem comprises of a SmartThings cloud, SmartThings hub, SmartThings mobile user app, and IoT smart devices. The applications implemented using SmartThings platform allow users to manage and control their home appliances (smart devices) via smart phones (mobile user app). The hub [23] acts as a gateway between the home IoT devices and the cloud services by connecting to the Internet. It supports several communication protocols including ZigBee, Z-Wave. All communications to and from hub to the cloud are encrypted using SSL (Secure Socket Layer)/TLS (Transport Layer Security) protocol. To execute cloud-based services, the hub must be connected to the SmartThings cloud. Without Internet connection, some of the services that are provided by the devices can continue (e.g. the light will stay on) but some services may become unavailable (e.g. sensing data and sending to cloud). Moreover,to change the settings of the devices (e.g. put them in low energy consumption mode) the hub needs to be connected to the cloud.

Apple HomeKit [12] is an IoT framework facilitates the process of managing and controlling connected IoT devices in a user's home via iOS device apps (e.g. iPhone - Home smart app). An iOS device (e.g. iPhone) that is connected to home network (e.g. WiFi) acts as a hub to communicate with connected devices. iCloud [10] is a common database for all connected devices and iOS devices. To provide remote control of the home, either Apple TV or an iOS device in the home and connected to the home WiFi network, will be used as a hub [11]. When the iCloud is not available, the HomeKit stops communicating with cloud. The devices may preserve their latest state before they disconnect with the iCloud and continue till they are connected again to the iCloud and updated.

Amazon AWS (Amazon Web Services) IoT [8] is a cloud platform for the Internet of things. AWS IoT enables internet-connected devices to connect and interact with the AWS cloud and other connected devices using MQTT (Message Queuing Telemetry Transport) protocol [7], using a Device Gateway that acts as an intermediary between connected devices and the AWS cloud services. In AWS IoT, applications communicate with device shadows that are virtual images of the connected devices. Without Internet connection, device shadows cannot receive new data from devices, but will continue communication with applications. Devices can continue functioning with their current states and will receive the updates from AWS IoT when the Internet is back on [7].

IBM Watson IoT Platform [18] on the IBM Cloud provides a framework for easily connecting devices to the IBM Cloud environment and manages them. It gives user a versatile toolkit that includes gateway devices, device management, and powerful application access. The platform acts as a hub for an IBM IoT ecosystem. A user can setup and manage their connected devices, and using their mobile apps, access the devices realtime or historical data, and perform analytics on them [17]. To connect devices and applications, the platform uses MQTT protocol and TLS, to securely receive data from and send commands to home devices. To access and change functionalities, devices must be connected to the cloud.

Cloud can become unavailable due to a variety of reasons, including:

• *Technical problems.* Network problems or temporary cloud shutdown due to technical problems or attacks (e.g. Distributed Denial of Service).
• *Natural Causes* such as storm, flood, fire or any extreme condition can leave the home without power or connection to cloud.
• *Scheduled maintenance* can make the cloud unavailable.
• *Targeted Attacks.* Malicious activities of attackers, who can be insider or outsider to the home, can make the cloud services unavailable.

The effect of cloud disconnection can be varied and includes:

• *Break-down of Home Security Systems.* Security monitoring systems (camera, motion detect, etc) and remote-controlled smart locks may stop correct functioning (e.g. a smart lock may keep the door open or locked and does not allow user control). These are essential functions at all times, in particular during emergency situations.
• *Providing Attacking Windows.* An adversary may use the home disconnection time for malicious tampering with the home computing system to plan their future attacks: an attacker may inject malicious code into the system, or tamper with the system configurations to provide future unauthorized access for insiders or people from outside.
• *Un-authorized access.* Incidental users of the smart home are those who are not involved in setup and configuration of smart home infrastructure. Such users, if malicious, may attempt to disconnect a smart home to remove the distinction between themselves and the primary users if the home access control is lost. This could enable them to have physical access to privileged spaces (private rooms) or devices (e.g. setup hidden camera or voice recording). A malicious incidental user may also try to observe a primary user's activities closely to gain future access when the security systems are down.
• *Safety.* A smart home can have a range of "smart" sensors for automation of smart appliances, or support for home assisted living. Malfunctioning of these sensors due to cloud disconnection can jeopardize safety of home residents. Below are important types of sensors that will be affected.
– *Smart Appliances.* Cloud-based smart home appliances receive a range of commands from cloud. Depending on the state of the device and the timing of the command, cloud disconnection could result in unsafe or inconvenient environment for home users (e.g. a thermostat may leave users in cold, or a smart light may stay on all night).

– *Wearables and home-assisted living.* There are a wide range of wearable sensors that are used for daily measurements of activities and in support of health related services such as reporting falls or other mishaps. Depending on the type of sensors and the duration of disconnection, the effect on the well-being of home users can be enormous.

– *Safety Sensors.* A range of sensors such as smoke and leak detectors have direct significance on the environmental safety: delaying of an auto-call to Fire Department will pose a severe risk to home safety.

3 RESILIENT SMART HOME

Motivated from the above discussion, our goal is to design an infrastructure for smart home that allows home to continues functioning safely for users when cloud is not available. As noted earlier existing smart home platforms rely on cloud to various levels. However in all cases reconfiguring device status (remotely) requires cloud connectivity. This reconfiguration is essential to save energy and put devices in emergency mode. As the first step, in the following section we provide a broad categorization of smart home services. This categorization can be modified or refined according to the specific needs of a smart home.

Home Service Classification. Table 1 shows typical essential services that a smart home may provide. Non-essential services would include automation of non-vital appliances such as a washing machine, or un-necessary functionalities such as different hues of light bulbs, or functionalities of an entertainment systems. This classification may vary depending on the preference of the user and other provisions such as the specifications of the backup generator and power supply.

Table 1: Essential services in Smart Homes

Type of Service	Function	Operation	Example
Connectivity (Internet)	Notification	Sends notification to user's mobile device on home status	Fire alarm is on or Cloud connection lost
	Remote Access	Allow a user to directly control home devices remotely	Open the door or view the live camera
Health Care	Patient Monitoring	Tracking patient vital signs and health status indicators using data collected from smart wearables	Smart device attached to user for pain treatment. Keeping track of heart rate, breathing, temperature, steps, detect body position in case if a person falls
	Remote Doctors	Able to read the victim's health information in real time and give instructions	In case the emergency medical services need instruction related to patient's health
Security	Smart Lock	Auto open or close	On smoke alarm, auto open door
	Smart Monitoring	Motion detection and Live camera	View live camera feed
	Smart Alarm	Sound alert, notify user, security patrol or police	Send notifications to user
Safety	Smoke Detector	Sound alert, run emergency automation, auto calling	Auto calling Fire, EMS and police
	Leak Detector	Sound alert, run emergency automation, auto calling	Alerting user or residents, calling security and fire
Reliability	Local Automations	Able to execute emergency routines	Auto calling authorities during emergency (e.g. fire)
Storage	Local Storage	Keep track of important data	Saving front door camera feed

RES-Hub. It is a stand-alone hub, that is powered by an alternative power generator if the main power system is unavailable, and has the role of maintaining the home functionalities. To ensure the functionalities of the smart home, albeit at a more basic level, RES-Hub must be able to do the following.

– Detection of cloud unavailability;
– Notification and ability to communicate (e.g. send status and receive commands) to outside world through alternate channels.
– Service transfer and ability to take-over the basic functionality;
– Transfer control back to the cloud.

In the following we briefly describe each of these functionalities.

• *Detecting Cloud Failure.* RES-Hub must detect cloud failure. Some possible methods are below.

– RES-Hub sends regular heartbeat to the cloud and receives the response. If the response is not received over a period of t seconds, it declares disconnection.

– There is no heartbeat but there is an expectation that a message will be received by a device in the house within a certain time period. This can be done by constructing a profile of the traffic that is received by the home, and using it to detect cloud disconnection.

– If the home uses a local hub, RES-Hub can interact with the hub or check the hub status to determine if devices are in normal communication with the cloud. – Additional methods such as checking Internet connectivity (for example through a ping command) or checking power outage can also be used in detection of cloud unavailability.

• *Notification.* Once a cloud disconnection is detected, a notification will be sent to outside world. For simplicity we assume this notification is sent to the primary home user. This requires RES-Hub to have access to an alternative mode of communication, for example a cell network.

• *Service Transfer.* When cloud disconnection is detected, RES-Hub must take over the role of the cloud, and when the cloud is back, it must transfer back the control. For this to happen, RES-Hub (i) must be able to communicate with the devices, and the cloud, (ii) have access to their status and setting before the disconnection, and (iii) have a description of services that must be offered, and (iv) the required software to achieve it. This means that the RES-Hub must be able to run the required applications and communicate with the devices, as well with the home owner (primary user) mobile device. It also implies that RES-Hub must obtain device status and setting at regular intervals. We assume a cooperative architecture where the cloud provider is willing to provide the required information to the RES-Hub (One may also consider non-cooperative architectures where RES-Hub may attempt to infer this information from communications of the devices to and from hub). Finally, the essential services and the their specifications must be provided to RES-Hub as part of its configuration. RES-Hub will transfer back services to cloud (including device status and history), when notified by the primary user. One may consider an automated transfer back, where RES-Hub autonomously detects cloud availability and transfer back the services. This adds to the complexity of the system and will be a future extension.

An important component of the RES-Hub is secure authentication, authorization and access control. As noted earlier, cloud disconnection can provide a window for attackers' un-authorized access. Thus RES-Hub must be able to authenticate requests and issue commands that are verifiable by the end devices. This is because during cloud unavailability a stricter set of security requirements will be necessary.

Other Requirements. Ideally we would like RES-Hub to be cloud neutral and support different home platforms. Our design of RES-Hub is based on IoTivity framework and can work with major cloud providers' frameworks including IBM, Apple HomeKit, Amazon AWS and Samsung SmartThings.

3.1 RES-Hub

Figure 1 depicts a cloud-based smart home equipped with RES-Hub. RES-Hub has access to a cellular network and a specification of authorized users and services. This information are stored in two tables: the *Registered user* table and appropriate authentication method, and the *Authorization* table that specifies accesses of authenticated entities in the system. These tables must be kept up to date, stored securely at RES-Hub. Once cloud disconnection is detected, RES-Hub will notify the home owner using a SMS over cellular network, and takes over the control of the home. RES-Hub and essential services are connected to a backup power unit (for e.g., a UPS system).

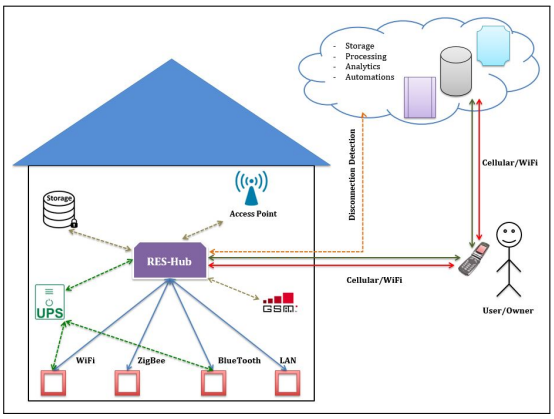

Figure 1: A Smart home equipped with a RES-Hub

4 RES-HUB DESIGN USING IOTIVITY

IoTivity [19] is an open source project for the Internet of Things (IoT) that is hosted by the Linux Foundation, and is sponsored by the Open Connectivity Foundation (OCF). OCF is a group of technology companies such as Samsung Electronics and Intel with the aim of developing standard specifications and promoting interoperability guidelines. RES-Hub's software design is based on Iotivity framework and is shown in figure 2. The framework allows devices to communicate with cloud regardless of manufacturer, operating system, chipset or physical transport [15]. Although one of the key technologies of connectivity is the constrained Application Protocol (CoAP) over UDP/IP, in this work, we will use MQTT (Message Queuing Telemetry Transport [14])over TCP/IP for connectivity between the devices and the hub.

Following the proposed architecture in [13] RES-Hub consists of three conceptual layers: 1: Network of Things Layer (NTL), 2: Common Service Layer (CSL) and 3: Application Layer (AL) with the following functionalities.

Network of Things Layer. This layer consists of two sub layers:

• *Network Connectivity.* This layer provides device discovery and support for secure communication over physical layer that can be

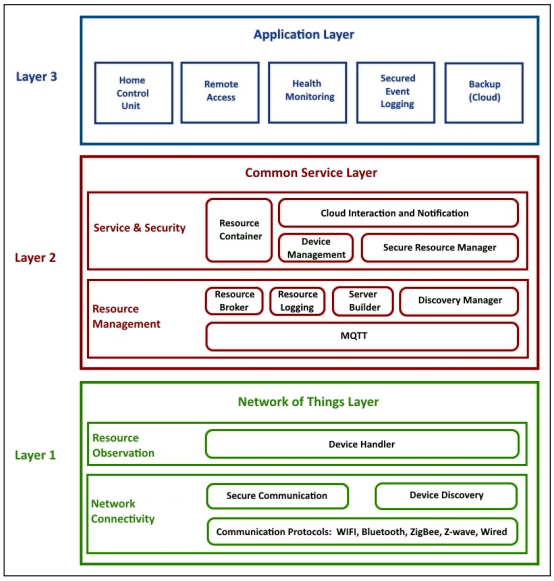

Figure 2: RES-Hub Software Architecture

wired and wireless channels (e.g. Bluetooth, ZigBee, Z-Wave). Example device discovery protocol is SSDP (Simple Service Discovery Protocol) [1] that is used to find devices on the home network, and serves as the backbone of Universal Plug and Play (UPnP) networking protocols. Secure communication module includes cryptographic libraries and support for TLS and establishing secure channel between two parties.

• *Resource Observation.* This layer has a Device Handler module. Device Handler represents a physical device and is the interface between the actual device and the Common Service Layer (CSL). By integrating MQTT and Server Builder in the Common Service Layer, a Device Handler can directly receive data from a device and send control messages to the devices (e.g., On/Off lights).

Common Service Layer. This layer has two sub layers:

• *Resource Management.* This layer provides the following modules.

– A Resource Broker that monitors the presence status of the user specified devices and provides the resource (device) status to an application.

– Resource Logging collects the data generated by the device during cloud unavailability period, and stores them in the Resource Container (database).

– Server Builder handles the simplified creation of resources, set the resource properties (e.g Resource type: temperature sensor) and attributes (e.g Temperature value).

– Discovery Manager manages all discovered devices (things) by the NTL.

– MQTT is a lightweight publish-subscribe based messaging protocol designed for machine-to-machine communication [14]. It is designed to work with constrained devices and low-bandwidth, unreliable networks like IoT and is used by Amazon AWS IoT and IBM Watson IoT.

• *Service & Security.* This layer will include the following functionalities.

– Secure Resource Manager manages Credentials, Authentication (including device IDs and credentials) and Authorization tables

(including Access Control list and/or capability. It is responsible for authenticating client and authorization. Submodules such as Policy Engine and Resource owner work together to provide access control in RES-hub.

– Resource Container[2], is to provide APIs for integration of non-OCF protocols into the OCF ecosystem [15].

– Device Management handles network connection setting and remote monitoring/reset/reboot functions.

– Cloud Interaction and Notification is responsible for interaction with the Cloud and managing notifications to the user and devices. The Notification module consists of, Presence Detection, Remote Request Handler, Resource State Synchronization components, that are used to monitor Cloud connection state, as well as receive status information from devices.

Application Layer. Applications access services that are offered by the Common Service Layer (CSL), through their application programming interfaces. Typical applications include health related application, home automation services, monitoring and billing. These applications receive data from the CSL and perform the required action as specified in its emergency specification.

4.1 Security

RES-Hub must ensure that the home functions as specified, when cloud is unavailable. Here we focus on the security provisions for this period, only. As noted earlier cloud unavailability marks an emergency period and so strict security design is necessary.

RES-Hub will use cryptographic authentication for users and devices, and OAuth based authorization for ensuring secure access to, and communication with, devices are maintained during cloud unavailability. Using cryptographic authentication would restrict the devices that can stay active during downtime. This is a design decision to protect against malicious attacks and can be replaced by other strong authentication mechanisms. Each user and device that interacts with RES-Hub will have a private key and a certified public key. Public key certification can be performed by the home owner who would act as the certificate authority (CA) for the device registration to RES-Hub . RES-Hub will safely store the public key of the CA which will be later used for certificate verification.

OAuth 2.0 [21][16] is a framework for distributed authorization. Entities can take the role of *Resource Owner (RO)*, *Client*, *Resource Server (RS)*, and *Authorization Server (AS)*. System *resources* are served by the RS. To access a resource, a *client*, after authenticating itself, must obtain an *access token* from the AS. In RES-Hub, MQTT protocol is used for communication of devices with RES-Hub, as well as services communicating with the devices. MQTT is run over secure transport layer, using protocols such as SSL/TLS or DTLS. MQTT broker must enforce the granted permissions in the token. A client who needs to access a resource will present the required access token to the broker. The access token is obtained by the client after authenticating itself to the authorization server that works in conjunction with the resource owner to generate the required access token. Once the token is obtained, it will present to the broker and the access will be granted. The role of RS will be given to the MQTT broker. Following the modular structure of IoTivity framework, the module responsible for authenticating client and providing access token is the *Resource Manager* that includes the submodules *Policy Engine (PE)* and , in our design, *Resource Owner(RO).*

Figure 3 shows the sequence of events when a client wants to subscribe to a topic (e.g. myhome/temperature) to receive updated messages (e.g. temperature information). The (MQTT) client

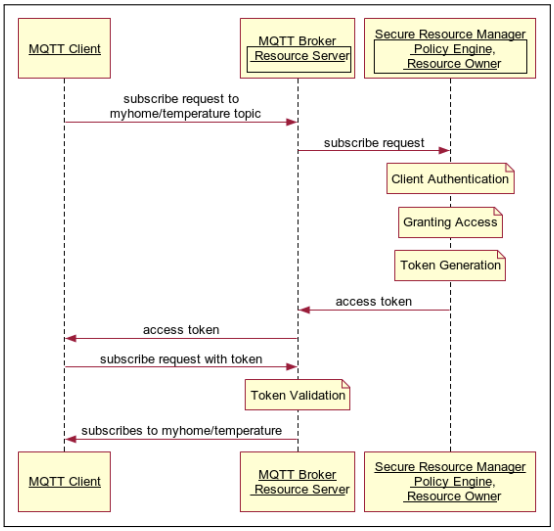

Figure 3: Resource access procedure in RES-Hub

presents its credential to the MQTT broker who forwards it to the Secure Resource Manager.The client will be authenticated by the RO and PE, and after gaining authorization, the required access token will be issued to the client. Client will present the token to the MQTT broker (who will act as the RS), and will allow the access. The token will be used by the client to access the messages of the topic that is subscribed (i.e. temperature). Broker may cache a permission list for validated token to improve overall performance.

Protection. Assuming secure communication and correct implementation, OAuth authentication and authorization will ensure access controls are appropriately granted and will be according to the specified policies. To ensure token authenticity and integrity, each token will be signed by the issuer and will be bound to the client identity. RES-Hub must be safe guarded physically (i.e. in a protected room) and must be implemented using appropriate security technologies (e.g. Hardware Security Module or trusted components) for secure implementation of software. The device must securely store its credentials, other secrets and policy tables.

5 A PROOF OF CONCEPT IMPLEMENTATION

For a concrete implementation of RES-Hub we considered Samsung SmartThings platform, a hub based architecture that relies on cloud for all the processing and generation of control commands. SmartThings Hub V.2 is planned to allow some local computation with the goal of service continuity when the cloud is down. The planned API will allow programming apps for the hub. These functionalities however are not fully implemented. Our implementation has been focused on two modules: Resource state synchronization, that gathers resource (device) state information from cloud at regular intervals. This information will be used by RES-Hub to provide uninterrupted services specified by user when the cloud is disconnected. Notification module, that is responsible for notifying the

user (e.g. by sending a SMS) when the cloud is detected offline. It can also notify other authorities if needed.

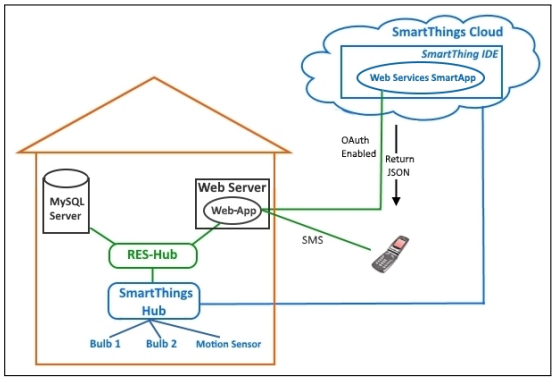

Figure 4: Resource state synchronization and Notification modules.

Resource state synchronization module. We implemented a Web Services SmartApp, Web Server and Web-App to synchronize resource state, which is depicted in figure 4. The Web Services SmartApp is developed in the SmartThings IDE user account and exposes the API endpoints that allow the Web-App to control a device, the device needs to register and pair with this SmartApp. The Web Server is implemented on a Raspberry PI.hub that simulates the RES-Hub. The Web-App runs on localhost that simulates the RES-Hub and makes calls to the SmartApp endpoint in the SmartThings Cloud for specific request such as get device status, etc. The collected data is stored in a local (home) database that is implemented as a MySQL database [4].

We use SamSung SmartThings Hub Version 2 with Motion Sensor and two Sengled Smart Connected LED Bulbs. The Web-App was developed using PHP [5], with Apache HTTP Server 2.4.29 [9] which is an open-source cross platform Web Server.

• *Authentication and authorization* . All SmartApps APIs are authenticated using OAuth 2.0 authorization protocol [21] to ensure that the Web-App has the requested access to the devices. We create Web Services SmartApp in SmartThings IDE with OAuth enabled. The SmartApp auto generates the OAuth Client ID and Secret that will be used when Web-App send HTTP request to SmartThings login page with OAuth enabled login. When successful logged in, a subsequent page is displayed that allows Web-App to authorize the devices this Web Services SmartApp can work with. SmartThings redirects back to our Web-App, it passes a code parameter on the URL. The Web-App uses this code (along with the OAuth Client ID and Secret), to get the token using the OAuth2 module, and store it in the session (e.g API Token: "*bbffef83-ea4e-479f-a567-90db62ee83e6*"). The Web-App uses this OAuth Token to make a *GET* request to the SmartThings API endpoints service at "*https://graph. api.smartthings.com/api/smartapps/endpoints*". The response is JSON [3] that contains the endpoint of our Web Services SmartApp (e.g API Endpoint: "*https://graph.api.smartthings.com/api/ smartapps/ installations/ a3310289-17a8-4069-ac7c-a6d94714f40a*"). Finally, Web-App uses OAuth Token and specific endpoint to make API call to Web Services SmartApp to request device status [24]..

• *Local database.* The SmartApp will respond in JSON format [3]. The Web-App will analyze the data, add time-stamp and store it to MySQL database, an example status message : "1, GR Mo, Detected Motion, Living Room, 2018-01-30 15:29:47.410453" (see figure 5).

Notification module. While Web-App sends HTTP request to SmartThings login page, Web-App also implements a PHP connection-handling function [6] to keep track of the SmartThings Cloud status. When the Cloud is unavailable, Web-App will send SMS (Short Message Service) through HTTP to user's mobile phone for notification.

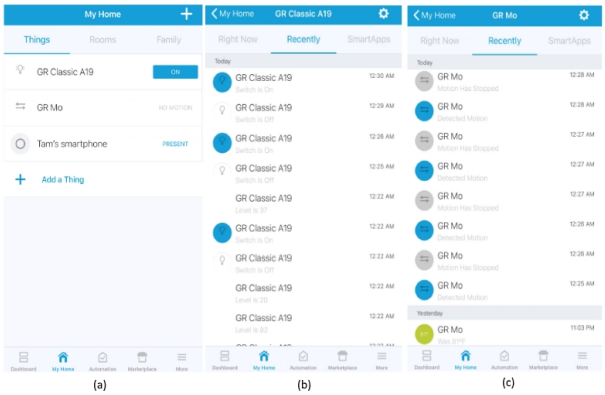

Figure 5: Demo of a SmartThings Mobile App: a) Home screen of the App; b) GR Classic A19 bulb history event; c) GR Motion sensor history event.

6 CONCLUDING REMARKS

Existing smart home platforms are primarily cloud-based. We raised the question of home resiliency when cloud becomes unavailable. This can happen because of unintentional causes, as well as through adversarial activities. We showed there are essential services that must be continued when the cloud becomes unavailable, and noted the need for providing a secure design that ensures cloud unavailability cannot provide a window of opportunity for attackers to gain immediate unauthorized access, or plan their future attacks. A very important aspect of cloud unavailability in existing cloud based platforms is the loss of control to re-configure devices. This is necessary to select the services that are essential and maintain their operation, while turning off un-necessary devices and service. We proposed RES-Hub to ensure the required functionalities are provided even when the cloud is unavailable. RES-Hub uses the most recent state of the devices and services (obtained from the cloud during normal functioning) to continue their operation according to the user specification, when cloud is unavailable. Based on IoTivity framework, we proposed a software architecture for RES-Hub that incorporates OAuth 2.0 authentication and authorization framework to guarantee secure access and control of home devices and services, when the cloud is unavailable. We also present a proof of concept implementation of SmartThings devices.

Our work is a first step in providing resiliency for smart homes: a topic which requires further research particularly as more devices and services are integrated into the home. It also calls for collaboration with smart home providers for refined APIs and access to their systems.

Acknowledgement. This research is in part supported by Alberta Innovates in the Province of Alberta, Canada.

REFERENCES

[1] 1999. Simple Service Discovery Protocol. https://tools.ietf.org/html/draft-cai-ssdp-v1-03. online accessed: March, 2018.
[2] 2018. IoTivity Resource Container. https://wiki.iotivity.org/resource_container. online accessed: May, 2018.
[3] 2018. JavaScript Object Notation. https://www.json.org/. online accessed: March, 2018.
[4] 2018. MySQL. https://www.mysql.com/. online accessed: March, 2018.
[5] 2018. PHP. http://php.net/. online accessed: March, 2018.
[6] 2018. PHP Connection Handling. http://php.net/manual/en/features.connection-handling.php/. online accessed: March, 2018.
[7] Amazon. 2018. AWS Developer Guide. https://docs.aws.amazon.com/iot/latest/developerguide/aws-iot-how-it-works.html. online accessed: March, 2018.
[8] Amazon. 2018. AWS IoT Framework. https://aws.amazon.com/iot. online accessed: March, 2018.
[9] Apache. 2018. Apache HTTP Server Project. https://httpd.apache.org/. online accessed: March, 2018.
[10] Apple. 2018. Apple iCloud. http://www.apple.com/lae/icloud/. online accessed: March, 2018.
[11] Apple. 2018. Homekit developer guide. https://developer.apple.com/library/content/documentation/NetworkingInternet/Conceptual/HomeKitDeveloperGuide/Introduction/Introduction.html#//apple_ref/doc/uid/TP40015050. online accessed: March, 2018.
[12] Apple. 2018. The smart home just got smarter. http://www.apple.com/ios/home/. online accessed: March, 2018.
[13] Sepideh Avizheh, Tam Thanh Doan, Xi Liu, and Reihaneh Safavi-Naini. 2017. A Secure Event Logging System for Smart Homes. In *IoT S&P@CCS. doi: 10.1145/3139937.3139945.* ACM, 37–42.
[14] Andrew Banks and Rahul Gupta. 2014. MQTT Version 3.1. 1. *OASIS standard* 29 (2014).
[15] Open Connectivity Foundation. 2016. About open connectivity foundation. https://openconnectivity.org/foundation. online accessed: March, 2018.
[16] Dick Hardt. 2012. The OAuth 2.0 authorization framework. https://tools.ietf.org/html/rfc6749. online accessed: March, 2018.
[17] IBM. 2018. IBM Watson Internet of Things educator guide. https://developer.ibm.com/academic/resources/internet-of-things-educator-guide/. online accessed: March, 2018.
[18] IBM. 2018. IBM Watson IoT Platform. https://internetofthings.ibmcloud.com/#/. online accessed: March, 2018.
[19] IoTivity. 2018. IoTivity Framework. https://www.iotivity.org/. online accessed: March, 2018.
[20] Microsoft. 2018. Microsoft Azure Suite. https://azure.microsoft.com/. online accessed: March, 2018.
[21] Microsoft. 2018. The OAuth 2.0 authorization protocol. https://oauth.net/2/. online accessed: March, 2018.
[22] Samsung. 2018. SmartThings Documentation. http://docs.smartthings.com/en/latest/. online accessed: March, 2018.
[23] Samsung. 2018. SmartThings Hub. https://www.samsung.com/us/smart-home/smartthings/. online accessed: March, 2018.
[24] Samsung. 2018. Web Services SmartApps. http://docs.smartthings.com/en/latest/smartapp-web-services-developers-guide/overview.html. online accessed: March, 2018.