smartHome

Til Blechschmidt
Student, Otto-Hahn-Gymnasium Geesthacht, Germany

1. Januar 2017

Inhaltsverzeichnis

Abbildungsverzeichnis

Tabellenverzeichnis

1 Philosophy

This project aims to provide an approach for connecting IoT devices that differs from current implementations. It attempts to do so in a decentralized and non-persistent yet robust fashion. One goal of this project is to create a network which is not flawed if a single device dies and simply adapts new devices without much configuration effort but still provides the possibilities to follow complex rules to trigger various actions based on changes in the network.

1.1 States

Despite the centralized persistent nature of other smart device networks out there, this implementation is based on so called states. A device may enter or leave a state and broadcasts this change of state to the network. Another device may then react to this change of state on their behalf. Take for example a car's Bluetooth that enters the area directly in front of the garage door. It broadcasts this state of being near the garage door to the network. Now the garage door actuator receives this change of state and reacts to it by a set of rules that are defined in its internal storage/configuration and for example, opens the garage door for the car to enter it.

Resistance against failures This way of handling events is more robust compared to a centralized set of rules since it doesn't rely on a single device that calculates reactions to events based on rules which might fail at some point rendering the whole network useless. If a device fails in this architecture it would only render that specific set of rules useless which has an overall smaller impact since chances are high that the real-world device the rules apply to has failed as well and the remaining network stays intact and operational.

Status broadcasts Whilst regular centralized networks make use of a central registry this is not the case with this approach. Instead every device stores its state internally. This opens up the possibility to differentiate between write requests to a specific channel or device and the results it causes. Take a write request to a whole channel as an example. There might be multiple devices listening to the channel and a single write request may result in multiple changes in state. By each device receiving the write request and processing it independently, each broadcasting the following change in state every other device in the whole network instantly receives a whole set of the resulting states caused by this single write request. This opens up the possibility for each device to process a write request differently in its own manner.

Read requests Take the case of a mobile device that may enter or leave the network at unknown times. Chances are that the network changes in the period this device is absent. This by itself would cause a huge headache since all changes would have to be synchronized upon reconnection. For most mobile devices it is the case that they do not require all states of every single device in the network at a given point in time. Because of this there will be no synchronization of states upon reconnect in the protocol itself. It might be implemented by a device that stores all states and then enables a mobile device to fetch this registry but it is not defined in the protocol itself. Instead the protocol defines read requests which can be sent to channels or devices which in turn return their current state through a UDP datagram back to the device which sent the read request. This enables mobile devices to fetch the states of specific devices at the time they are required reducing the load and overhead.

2 Channels

A channel describes a group or collection of Devices. A channel by itself is nothing more than an ID in the heap of some devices. It can be referenced to by a string defined by the user such as 'Living Room' or 'Kitchen' by utilizing the Channel DNS stack. A group of devices listens to requests which are aimed at a specific channel and reacts to possibly but not exclusively only those.

2.1 Get list of devices in channel

3 Devices

3.1 Passive devices

A passive device consists of two parts. The software layer and the hardware layer both in conjunction enabling the network to interact with devices in the real world like lamps or TVs.

Software layer The software layer responds to requests coming in from the network. The responses or actions might be limited to a specific channel but can use various wildcards as well. That's up to the developer and use-case. Note that a software layer is NOT limited to representing one device. It is possible for a software layer to run multiple instances of a class that reacts to incoming messages. Therefore it is possible for one software layer to control multiple hardware layers like lamps for example.

Hardware layer The hardware layer represents the part of the program that takes commands from the software layer and translates them to either interactable items like a screen or trigger some actions in the real world like a switch to turn on a lamp.

3.2 Active devices

An active device is sending status updates and requests to other passive devices reacting to those. An example of an active device would be a smart-phone. It can enter a specific perimeter or receive a call and sends these changes in state to the network. Note however that these changes are not saved by any central authority. All devices that would react to the change in state do so upon reception of the datagram and do not store the state by itself. But it's possible for any 3rd party to add a device that stores these states for later access like showing them to a user that was disconnected from the network when the change in state took place.

4 Channel DNS

Since humans usually can't follow along with huge amounts of numbers the channel IDs used in the Communication protocol are translated to strings by a Channel DNS system. In its most basic form, this system passes around changes that were done to the registry including a timestamp. Every device in the network receives a copy of this registry and by adding up all changes it is able to resolve channel names to IDs.

Fetching full registry Since devices which just joined the network don't have the initial stack and would have to wait for a change to appear (which in turn would require the whole registry to be sent to every device) a device that joins a network asks for an up-to-date registry upon connection. This includes the problem that all devices would respond with the same data, creating unnecessary overhead. In order to prevent this problem the newly connected device asks waits for any arbitrary datagram to be transmitted across the multicast channel and then asks the sender of that datagram for the registry.

Incremental updates Because all devices have synced their registries on their first connect it is only required to send incremental updates. So if any device is requesting a change to a registry entry it broadcasts this change together with a timestamp to all connected devices which in turn insert this change into their registry.

Registry overflow As all changes are written into the registry rather than the current state it will overflow at some point. To counteract this problem all devices resolve their registries on receiving an update and delete all unnecessary entries like multiple consecutive renames or in the case of a deletion all previous changes to that entry.

5 Communication protocols

5.1 UDP

5.1.1 General structure

All UDP communication is split into two parts. One being direct communication between two devices following a special protocol that is dependent on the task and the other part being indirect communication via multicast sockets.

5.1.2 Multicast

The datagrams in the multicast channel are all JSON formatted. This JSON object may have the following keys:

channel	integer
action	string
payload	object

action may take one of the following values: read, write, state

5.1.3 Device scanning

5.1.4 Updates

5.1.5 Read & Write workflow

5.1.6 Channel DNS incremental synchronisation

5.2 TCP

5.2.1 Channel DNS registry synchronisation