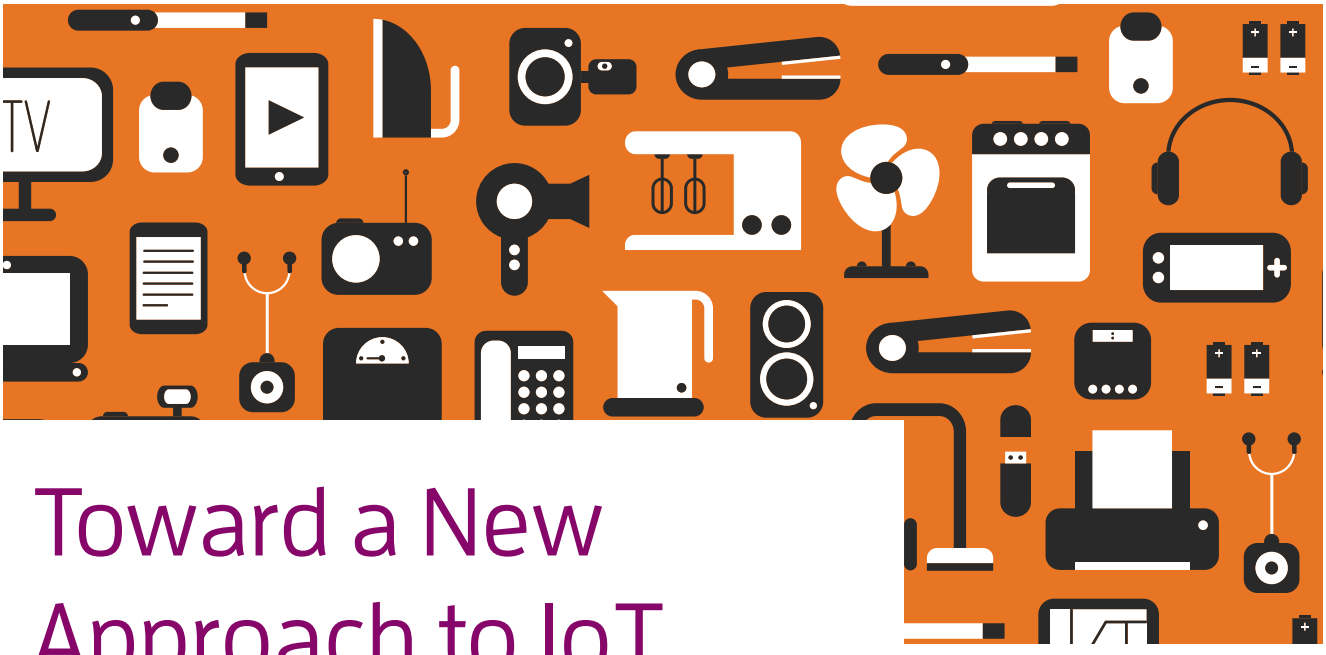

Toward a New Approach to IoT Fault Tolerance

Doug Terry, Samsung Research America

Although it offers effective fault tolerance, three-way replication triples a system's purchase price. The natural redundancy of functionality across devices within the home, as well as usage scenarios, should be exploited in a new approach to IoT fault tolerance.

FROM THE EDITOR

The Internet of Things (IoT) faces many challenges that have been impeding the definition of a unifying standard, including privacy and security guarantees for its users. An equally important issue is system reliability, which implies fault tolerance. If IoT systems aren't as reliable as the traditional systems they're replacing, they will fall short of user expectations. In this article, Doug Terry of Samsung Research America discusses the core issues that need to be addressed and some potential solutions. –*Roy Want*

Consider a simple smart-home application: when someone enters a room, the lights automatically turn on using a Web-based service, such as IFTTT (If This Then That). IFTTT allows users to create chains of simple conditional statements (called "recipes") that are triggered based on changes to environmental conditions or to other applications such as Twitter and Facebook. An application like this would require a sensor to detect when a person enters the room—this could be a motion sensor, such as the one sold by Samsung SmartThings—and a smart light bulb, such as a Philips Hue. This system would also require a hub for the motion sensor and a hub for the light bulb (as of this writing, the SmartThings and Hue hubs are incompatible).

The motion sensor and the smart light bulb communicate wirelessly with their respective hubs, perhaps using the wireless protocol ZigBee, and both hubs connect wirelessly to the home's Internet router, likely

0018-9162/16/$33.00 © 2016 IEEE

EDITOR **ROY WANT**
Google; roywant@google.com

using Wi-Fi. The application itself runs on the cloud, subscribes to events from the motion sensor (which are relayed via its hub), and sends commands to the light bulb via its hub. This configuration is depicted in Figure 1. Setting up such an application can be done in a matter of minutes, but what sort of fault tolerance would it provide?

A failure of any of the devices involved would render the application inoperable:

- the motion sensor could fail to detect a person, perhaps because the sensor's battery is dead, or it could detect a person but fail to inform the application due to communication outages;
- the light bulb could fail to turn on because it has burned out or lost the network connection to its hub;
- either hub could fail, cutting off communication to the attached devices;
- the Wi-Fi router could fail, preventing anything in the home from accessing the cloud where the application runs;
- the application itself could even fail—although most cloud providers offer fault-tolerant software platforms; or
- there could be correlated server failures or even entire datacenter failures.

Every stage of the application pipeline is a single point of failure, yet this is how smart apps run today. Clearly, it's desirable for the application to continue operating in the face of (limited) failures, which requires some redundancy.

THE TRADITIONAL APPROACH

The common approach to achieving fault tolerance—a subject that's been studied for decades—is to use triple

Figure 1. An example lighting application and its devices.

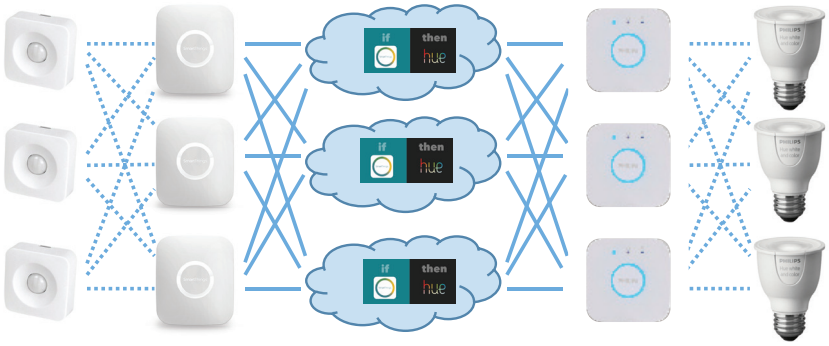

Figure 2. A configuration with triple modular redundancy.

modular redundancy and majority consensus. That is, we could build each software module as a replicated state machine with three instances running in parallel on independent hardware, with the output being the value agreed upon by a majority of instances. This allows the system to survive any single hardware or software fault.

In the case of our lighting application, this would mean using three motion sensors, three light bulbs, six hubs (three of each kind), three Internet routers, and possibly even three cloud providers running three instances of the application code. Plus, we would need some additional protocols and software for ensuring that events from different sensors are received in the same order at replicated components and for voting on the outputs of each component while eliminating redundant messages. Doesn't that sound a bit excessive?

Even with all that hardware, as shown in Figure 2, it would be difficult for a skilled programmer to build a fault-tolerant application. For one thing, existing Internet of Things (IoT) platforms have limitations; for example, IFTTT only supports recipes with a single "if" device and a single "then" action. SmartThings allows programmers to write arbitrary Groovy (www.groovy-lang.org) code that interacts with any number of devices, but each SmartThings device can only connect to one SmartThings hub. Some IoT systems currently only support a single hub per home. Beyond these limitations, the programmer still needs to write code that subscribes to multiple motion sensors, runs majority consensus on the events that are reported by these sensors, deals with failures that delay events or cause them to not be delivered, sends commands to multiple light bulbs, ensures that these commands are actually executed on one or more devices, and so on. There must be a better way.

A NEW APPROACH

The characteristics of smart-home environments, applications, and devices

suggest that other approaches to fault tolerance might be more practical. The most appropriate scheme for handling failures in an IoT system should exploit the following pertinent observations:

- *Devices are mostly fail-stop*. A motion sensor could experience a hardware failure that causes it to report motion when none exists, but that's not the typical failure mode. More likely, the motion sensor simply stops working. Addressing this sort of failure doesn't require three motion sensors or majority consensus. Two sensors are sufficient if, at most, one of them is down at any given time. Similarly, three light bulbs aren't necessary if the only requirement is to turn on one light, and it's reasonable to assume that a failed bulb is promptly replaced before a second bulb failure occurs.
- *Different types of devices can report similar events*. There are many ways to detect whether a person has entered a room; in addition to a motion sensor, a video camera is equally adept at observing motion. A microphone (embedded in a TV or Amazon Echo) that picks up sounds would also work, as would a door open/closed sensor. Of course, a smartphone attempting to connect to the in-room Wi-Fi router is also a sign that its owner is near. Rather than using three instances of the same type of motion sensor, the system could incorporate presence notifications from a variety of devices that might already exist in the home.
- *Hubs exist in different forms*. There's no need to purchase three instances of a special-purpose hub. As previously mentioned, two should suffice because hubs tend to exhibit fail-stop behavior. Moreover, hubs already exist in a variety of devices, such as smart TVs, refrigerators, Internet routers, and voice-activated assistants. Future smart homes are likely to contain multiple hubs that have different hardware characteristics and different failure properties. The system should take advantage of this existing redundancy.
- *Diverse wide-area networking technologies are available or pending*. Rather than relying on multiple Internet routers—which would be costly and possibly ineffective because they'd most likely share a single broadband connection from the home—network fault tolerance can be attained through the use of different types of wide-area networks. For example, cell phones communicate over cellular networks using connections that are completely independent from those established by Internet routers using DSL or cable modems. A cellular modem could easily be built into hubs to enable continuous communication with a cloud provider in the face of intermittent Internet connectivity. New low-power wide-area network (LPWAN) technologies in support of IoT applications, such as LoRaWAN (www.lora-alliance.org), are also on the horizon.
- *Smart apps are often stateless event handlers*. IoT applications subscribe to events, process the events, and take actions. Any local state is typically a soft state that can be reconstructed when the application is restarted, such as cached sensor values. Thus, there's no need for, and little benefit from, running a smart application as a replicated state machine.
- *Applications can respond leisurely to external events*. When processing an event, application delays of a second or so will often go unnoticed by humans. Although turning on a light when a person enters a room might happen within milliseconds, the person is unlikely to be surprised if there's an occasional delay. If an application becomes unreachable, perhaps due to a failure of the hub on which the application was running, there's ample time for other hubs to detect the failure and restart the application. The newly restarted application can resubscribe to external events and resume processing.

> IoT fault tolerance is imperative, as users expect their smart applications to operate correctly and continuously.

In light of these observations, one can imagine an IoT platform that provides application fault tolerance without resorting to active replication. Applications can be written in the usual manner and be made fault tolerant transparently by having the platform discover and select nearby devices that can report similar events, such as motion or presence, and that support similar actions, like turning on a light. Devices should automatically connect to nearby operating hubs and automatically switch hubs as failures occur. Such hubs should include multiple types of networks and use whatever connections are available. The platform should automatically detect

which devices are currently functioning and use them appropriately. Application state, if needed, could be stored persistently so that if there's a failure of the hub or server on which the application code is running, the application can recover from its previously saved state, but in most cases an application can be restarted in a new location with no persistent state.

OPEN ISSUES

Although transparent fault tolerance might work well for some IoT applications, it might not be ideal in all cases. For example, the lighting app might want to consider which sensors are reporting the presence of a person, along with the accuracy probability of these reports, so that it can evaluate which reports are best to act on. If the platform simply picked a single sensor to deliver events to the app, it might choose a motion sensor in the corner of the room rather than near the door, causing the application to miss the arrival of a person. Or worse, the system might inadvertently choose a sensor that's connected to a hub located in an adjoining room. Though application users might want the system to turn on a light if any sensors report the presence of a person, this could lead to occasional instances of a light being turned on incorrectly.

Similarly, applications might need to determine whether to send commands to one or a set of actuators. If a room has multiple lights that are functioning correctly, should the smart app turn on all the lights in that room when a person enters or just one? If just one, which light is preferred? Perhaps the user wants to specify a priority-ordered list of devices. How best to incorporate application-specific requirements and user preferences into an IoT fault-tolerance scheme is an interesting open issue.

Another concern is that current smart apps and their underlying platforms take a "fire-and-forget" approach when issuing commands to actuators. Such commands are generally sent asynchronously using best-effort delivery. If a light bulb has burned out, sending it a command to turn on the light will be ineffective. Or perhaps the command is lost in transmission although the light bulb is functioning properly. How does the platform discover this? From a control-theory perspective, IoT systems could learn the benefit of a closed-loop approach in which, based on observational feedback, commands are retried until the desired state is achieved. It might be possible to query a device's state to determine whether it has successfully processed a previous request, such as pinging the light bulb to determine if it's currently on. It might also be possible that other sensors, such as a photometer, could be used to discern the current state. Should detecting failed commands, and then recovering by reissuing them, be done automatically by the platform or by the application code? In general, the proper detection, and perhaps recovery, mechanism would appear to be device specific.

The platform might also need to identify whether commands are idempotent; that is, whether issuing a command more than once could lead to problems. In the case of a light bulb, turning on the light multiple times is perfectly acceptable. So, if there's ever any doubt whether a request to turn on the light has been performed or if the application fails while trying to issue the command, the platform or application can simply retransmit the request. Similar reasoning applies to commands that set a thermostat temperature or unlock a door. On the other hand, if the command is to water a plant, giving the plant twice as much water as it needs is detrimental. Thus, more thought is needed about the interplay between fault tolerance and application semantics in an IoT world.

IoT fault tolerance is imperative, as users expect their smart applications to operate correctly and continuously. Traditional techniques for achieving fault-tolerant program execution should be revisited in the context of smart homes. A key challenge is dealing with the tradeoff between cost and reliability in a world of devices with widely ranging characteristics including functionality, failure rates, recovery modes, and purchase prices.

Even human expectations for fault tolerance might vary across devices. When replacing simple mechanical devices (like a light switch) with electronic counterparts (like a motion-sensing smart app), humans will expect similar reliability. The minimal baseline for IoT fault tolerance should be based on common experience with physical objects in a non-IoT world. A light switch rarely fails to turn on the light that it controls. Thus, a smart app controlling the same light must also rarely fail if the new technology is to be accepted. On the other hand, people readily accept that light bulbs will occasionally burn out. Thus, although replicated applications might be desirable, redundant light bulbs might be less critical. IoT system designers can look for opportunities to improve the status quo, such as by predicting a light bulb's life span and automatically reordering a replacement bulb, without adding significant cost.

Cost is often cited as a key barrier to adoption of smart-home technologies. Although it offers effective fault tolerance, three-way replication triples the purchase price of a system, making such systems considerably less attractive for consumers. Thus, replication must be employed judiciously. The natural redundancy of functionality across devices within the home, as well as usage scenarios, should be exploited in a new approach to IoT fault tolerance.

DOUG TERRY is a Distinguished Research Engineer at Samsung Research America. Contact him at doug.terry@samsung.com.