Home, SafeHome: Ensuring a Safe and Reliable Home Using the Edge

Shegufta Bakht Ahsan[†], Rui Yang[†], Shadi A. Noghabi[*], Indranil Gupta[†]
{sbahsan2,ry2, indy}@illinois.edu[†], shadi@microsoft.com[*]
[†]*University of Illinois at Urbana Champaign.* [*]*Microsoft Research.*

Abstract

As smart home environments get more complex and denser, they are becoming harder to manage. We present our ongoing work on the design and implementation of "SafeHome", a system for management and coordination inside a smart home. SafeHome offers users and programmers the flexibility to specify safety properties in a declarative way and to specify routines of commands in an imperative way. SafeHome includes mechanisms which ensure that under concurrent routines and device failures, the smart home behavior is consistent (e.g., serializable) and safety properties are always guaranteed. SafeHome is intended to run on edge machines co-located with the smart home. Our design space opens the opportunity to borrow and adapt rich ideas and mechanisms from related areas such as databases and compilers.

1 Introduction

The smart home market is growing disruptively, and is predicted to balloon from the current $27B market size to $150B by 2024 [40, 41]. Simultaneously, both the density and diversity of IoT (Internet of Things) devices inside a smart home are increasing. An average home will contain over 50 smart devices by 2023 [15]. Roughly 1,500 vendors sell IoT devices [30], from giants (Google, Amazon) to startups [35]. Smart devices cover all aspects of the home, from safety (fire alarms, sensors, cameras), to doors+windows (e.g., automated shades), home+kitchen gadgets, HVAC+thermostats, lighting, garden sprinkler systems, home security, and others.

In today's smart homes, devices are controlled by *commands*. A command is a user-initiated or programmatic-triggered action that changes the state of an individual device, e.g., turn on kitchen light, open bedroom window, increase temperature to 73°F. Commands are usually issued through smart home controllers, e.g., Alexa, Google Home, Siri, SmartThings, etc. [2, 5, 16, 37]. To add convenience to the home automation, major smart home controllers extend the interaction with smart devices beyond merely single commands, to a sequence of commands, or so-called *routines* [3, 43]. Routines are needed for either convenience (e.g., turn on group of bedroom lights, switch on entertainment system), or for correct operation (e.g., run garden sprinklers in four house corners sequentially, 20 minutes each). Table 1 defines these common smart home terms and other important terms.

Today's commercial smart home systems, such as Alexa or Google Home, are *best-effort* systems. They have scant support for correctness, safety, or failures. Further, they have a strong cloud dependency, which means that under disconnection or cloud outage, the entire home system becomes unprotected and unavailable [31]. As a result, a major challenge with smart homes has become the *unsafe and unreliable states* they may go into, as seen by many user-reported incidents [25]. We observe that these unsafe states commonly arise from one or more of the following three main reasons: i) *concurrent* commands and routines, ii) *failures* and disconnections, and iii) *incorrect programming* of routines.

First, commands and routines, if executed in an ad-hoc fashion, may conflict with other concurrent commands or routines, thus, create an inconsistent state that affects the human users. For example, consider a "leave home" routine which turns off all devices including the stove and the fan. Alongside, consider a case where it is required for the kitchen exhaust fan to be on when cooking food, as the smoke and vapor might kill the pet birds [10,45]. If a person leaves home (turning off all devices) and simultaneously their spouse on their way home initiates a "cook food" routine from their phone (turning on the stove and exhaust fan), a possible end-result would be the stove on and the exhaust fan off, as there is no proper isolation among routines.

Second, the smart home is an unreliable environment with a collection of commodity devices and high chances of failures. Devices can fail at any time and the recovery time might be unpredictable [25], causing unsafe situations. For example, consider the "cook food" routine turning on the stove and exhaust fan. For safety, it is expected that if the stove is on, the fan must also be on. However, if the exhaust fan fails to turn on or fails afterward, it will violate the safety expectation.

Finally, a smart home includes a variety of devices supporting complex commands. Even without failures and conflicts, designing bug-free routines that do not create unsafe outcomes is not easy. A "leave home" routine programmed to lock the front door but forgetting to list a command to switch on a newly installed security alarm, is a violation of safety. Such situations can be complicated further by myriad possible interactions among routines (as shown by the earlier examples).

At the same time, users of smart homes are general public with little to no programming skills. Even for the tech-savvy user, reasoning about concurrency and failures is non-trivial (as every database developer knows!). It is inconceivable that future smart homes will continue to rely solely on human users to arbitrate such complex interactions.

Table 1: **Terminology used in smart homes.**

Term	Definition
device	a smart home device with a set of potential states
command	a user/program triggered instruction that changes the state of an individual device
routine	a sequence of commands
safety-properties	guaranteed device behaviors that user expects from the smart home.

What is needed is an integrated approach that abstracts away the complex parts of smart home management while continuing to offer users programmability and flexibility in defining routines and safety clauses.

A number of industrial and research systems have each addressed parts of the underlying complexities in smart homes. Some systems [14, 36] use priority-based approaches to address concurrent device access. Others [6] propose mechanisms to handle failures. A few systems [7, 28, 33] formally verify procedures. However, none address the entire picture.

We introduce *SafeHome*, an autonomous system for a safe and reliable smart home. SafeHome supports the full flexibility of being able to imperatively define complex routines (containing sequences of commands), while also guaranteeing a set of desired properties, called "safety properties", specified by the user in a declarative manner. Internally, SafeHome includes components for catching and responding to concurrency conflicts, safety violations, and failures. Each component is modular and transparent, i.e., it can easily be swapped with another state-of-the-art mechanism without any change in the user's routines. Finally, SafeHome encapsulates an edge-first philosophy to its design, assuring safety properties even under cloud disconnection. Our edge-based approach means that device firmware does not need to be changed, and that SafeHome interacts directly with APIs and commands supported by devices.

A few salient features of our approach are as follows. First, borrowing from the well-known and widely-tested notion of *ACID* properties in transactional databases [19], SafeHome proposes a new thought paradigm tailored to the smart home, which we call *SafeHome-ASID* (*A*tomicity, *S*afety, *I*solation, *D*urability). Second, borrowing from compilers, SafeHome's imperative routines undergo both static checking as well as dynamic checking, w.r.t. the declared safety properties. Finally in spite of the analogues, a smart home is different from a database, e.g., intermediate states of devices are seen by users in the former, and this entails subtle design considerations.

2 Issues in Today's Smart Homes

Human users have certain safety expectations from their smart home environments. These safety expectations may be life-critical (e.g., avoiding fire, power overload) or highly-desirable (e.g., reducing energy waste). Yet, today's systems are widely known [25] to repeatedly violate safety and reliability. We discuss the main reasons behind this status.

Table 2: **Unsafe States and Safety Properties to catch them.**

Undesirable State	Desirable Safety Property
Routine R1 turns on both stove and exhaust-fan, but then Routine R2 turns off exhaust-fan.	**if** `(stove==ON)` **then** `(exhaust-fan==ON)`
Routine R1 opens a window, Routine R2 turns on air-conditioner.	**if** `(air-cond==ON)` **then** `(windows==CLOSED)`
Subsequent commands switch on multiple power-hungry devices, causing power overload.	**if** `(dishwasher==on)` **then** `ATMOST(1)` `(washingmachine==ON, dryer==ON)`
Turning on all sprinklers around the house leads to insufficient water pressure.	`ATMOST(1)` `(Northeast-sprinkler=ON, Northwest-sprinkler=ON, Southeast-sprinkler=ON)`
User accidentally left the garage-door open overnight.	**if** `(garage-door.OPEN > 'n' hours)` **then** `(garage-door==CLOSE)`

A. Poor Programming of Routines: In its most generalized form, an assistant or user of a smart home executes a mini-program of commands. We call such mini-programs as *routines*–defined, in simple terms, as a sequence of commands (Table 1). A routine may be either user-initiated or trigger-based (time or event trigger), and either short (e.g., turn off all lights) or long (e.g., sprinkler system).

As smart homes grow in complexity, designing routines to control the home also becomes an increasingly complicated and error-prone activity. A wrong *sequence* of commands/routines, *poorly designed* routines, or *ill-maintained* routines (especially when adding or removing devices, which is typical in a smart home) can put the smart home in an unsafe state. Table 2 (first column) shows several such examples.

This shows the need for: a) users to have a way in which to *specify safety properties*, as well as b) for underlying mechanisms in a smart home to *ensure safety in spite of concurrent execution*. Table 2 (second column) also shows the corresponding desirable safety guarantees for each example.

B. Concurrency Conflicts: Smart home systems have been designed with the assumption of "running one command at a time". However, a smart home can receive commands concurrently from many sources: multiple users, multiple assistants in different rooms, and pre-scheduled or sensor-triggered commands. Executing concurrent and overlapping routines might leave the smart home in an *inconsistent state*, i.e., a state with a mixture of the effect of two (or more) concurrent routines. This is not addressed by today's smart home systems.

We measure the likelihood of such inconsistent states for a common setup. We run two concurrent routines on seven TP-Link HS105 [46] smart plugs. Routine R1 turns on all lights sequentially, when routine R2 turns off all lights after a minuscule interval. There are two desirable outcomes from this concurrent execution–all lights are off, or all lights are on. However, Figure 1 shows that as the number of devices touched by each routine rises, the probability of mixture result (some lights on with others off) grows quickly. Further, if R2 starts sooner after R1 (different lines in the plot), the chances of inconsistency become higher.

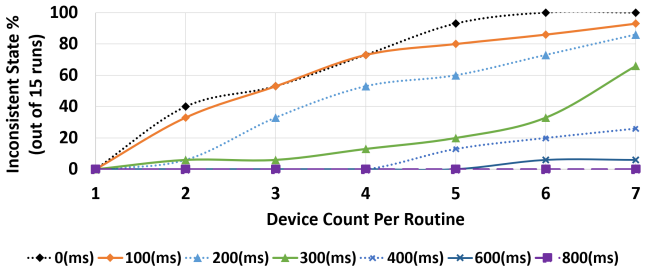

Figure 1: **Concurrency results in conflicts in a real smart home deployment.** *Two routines R1 (turn on all lights) and R2 (turn off all lights) executed on a varying number of devices (x axis), and with routine R2 starting a little after R1 (different lines). Y axis shows fraction of end states that are not serialized (all off, or all on).*

This shows the need for mechanisms in the smart home which can ensure that routines execute *atomically* and in an *isolated* way from each other.

C. Unpredictable Failures: Like other commodity machines, smart IoT devices can also fail unpredictably. Today, there is little failure support available in a smart home environment, with only a "best-effort" philosophy being prevalent [31].

Failures can negatively impact both: a) execution of routines (e.g., not turning on all lights when a light is failed), and b) safety property violations (e.g., exhaust fan failing while the stove is still on).

This shows the need for mechanisms that *detect device failures*, and policy and mechanisms that *react to such failures*.

D. Strong Cloud Dependency: The popular home automation systems including Google Home [16], Alexa [2], SmartThings [37] rely heavily on their own cloud infrastructure [48, 49]. This reliance can break during routine network outages [12, 51], or due to (lower but non-negligible probability) cloud outages, with examples abounding from Google Home [17], Alexa [1], and others [18].

In all these cases, failures disrupt the home-automation system's availability [4, 20, 38]. Even if there were a safety mechanism in place in these home-automation systems, this mechanism (in the cloud) would be unavailable to the smart home devices under disconnections. If a critical home device (e.g., fire-alarm) fails during this period, it will go undetected, even though the smart home has plenty of compute power available onsite including the smart home controller.

This shows the need to *house much of the logic for concurrency and safety in the edge*, in (or nearby) the smart home.

3 SafeHome's Design: Salient Aspects

The goal of our system, named *SafeHome*, is to achieve the vision of a reliable and safe home at all times, despite the challenges arising from concurrency, safety concerns, failures, and complex routines. SafeHome materializes the reliable and safe home vision using an edge-based architecture. This section describes key abstractions and mechanisms in SafeHome.

3.1 A New Abstraction: SafeHome-ASID

SafeHome aims to abstract away the burden of considering concurrency, failures, and safety away from the human user. Towards this, SafeHome introduces a new thought paradigm called *SafeHome-ASID*.

ASID stands for SafeHome versions of **A**tomicity, **S**afety, **I**solation and **D**urability, and we define these terms below. This is inspired by the well-known and well-tested ACID properties that transactional databases routinely offer. The analogy is apt because: a) in a smart home, commands change the state of devices, and a routine consists of a sequence of commands (Table 1), while b) in a transactional database, operations change the value of server objects, and a transaction consists (simply speaking) of a sequence of operations.

However, in spite of the analogy, a smart home is different from a database in many ways. As a result, we define new SafeHome-ASID properties for smart homes as follows.

- **A: SafeHome-Atomicity**. Execution of a routine is *atomic and exactly-once*. When a routine finishes, either: a) all its commands have been executed successfully, or b) none of its commands have had an effect on the smart home. Atomicity also implies that SafeHome will have to reason about what it means to abort a routine, and undo its operations.
- **S: SafeHome-Safety**. Safety properties are *specifiable* in a clear, declarative fashion, and concurrent routine execution *always satisfies all safety properties across the smart home*. We expand in the next subsection. Note that safety properties often span multiple devices (Table 2).
- **I: SafeHome-Isolation**. Concurrent routines are isolated from interfering with each other at devices. If they interfere, SafeHome will have to ensure the execution is *serially equivalent*, i.e., the end state of the smart home is equivalent to executing one routine at a time.
- **D: SafeHome-Durability**. A routine that completes successfully cannot be undone (except by another subsequent routine).

Satisfying these properties in SafeHome requires special care in design. Naively using mechanisms from the DB transactions world could make the home too restrictive–any device failure reverts the whole routine which is not always necessary, especially given that devices are not inherently highly-available. For example, a failed light should not prevent a "leave home" routine that locks the doors and turns off lights.

Hence, in SafeHome, we allow a subset of a routine's commands to be tagged as critical (by the programmer/user), with remaining as optional not requiring for successful routine completion (e.g., a "leave home" routine must lock the doors and close windows, and attempt to turn off lights).

3.2 Safety Properties

Specification: While safety properties can be specified in a myriad number of ways [28, 54], we find that a large majority

of clauses can be specified using the following grammar.

```
A:- if A then A else A
A:- DeviceID.StateID ==<>!= foo
A:- ALL(A), ANY(A), !A, ATLEAST(k)(A), ATMOST(k)(A),
  A AND A, A OR A
```

In SafeHome, a user may define a safety property either because it is critical to human safety, e.g., `if (stove==ON) then (fire-alarm==HEALTHY)`, or for user convenience, e.g., `if (GarageDoor.State==OPEN) then (GarageLight.State==ON)`. Further examples are shown in Table 2. We envision that some safety properties will come "baked" into the smart home, while others can be programmed and changed by the user. All safety properties admitted into SafeHome are stored in the edge, and continually checked.

Safety Checking of Routines: SafeHome performs safety checking at multiple levels:

1. **Static Safety Checking:** Does a routine by itself violate safety? This is a compile-time checker. (e.g., a routine that turns on the air conditioner and opens the window).

2. **Dynamic Safety Checking:** Does a routine violate safety, given the current state of the home (device states, failures, and other currently-executing routines)? This is a runtime checker. This checking can be done either *eagerly* (when the routine is started, check all its commands for safety), and/or *lazily* (for each individual command, right before it is initiated). While lazy checking is mandatory, eager checking can catch violations early and prevent wasted work. Both types can be used together.

Unexpected states such as failures, conflict resolution (e.g., aborting or reverting mid-routine), externally initiated changes (e.g., manually turning a device off), and certain sequence of routines, can lead a smart home to unsafe states. Dynamic checking identifies these situations.

3.3 Failure-Tolerance

Failures of devices need to be detected by using an appropriate failure detector protocol running across the devices and the edge. Failure detection, alongside the stateful nature of devices, is crucial in ensuring the durability property (D). We are currently exploring both fully-distributed and edge-based failure detectors. Both of these classes [13, 27, 44] rely on timeout mechanisms (heartbeats or pings).

SafeHome's response to a failure may affect other devices in a home (to satisfy safety properties), as well as currently-executing routines (for atomicity, isolation, and safety). When a safety property is violated, SafeHome offers the guarantee of a *tolerance window* within which all safety properties will again be satisfied. The tolerance window may be set either by the user (e.g., exhaust fan violations must be resolved within 5 seconds) or physical requirements (e.g., humidity sensor reboots within 3 seconds, so SafeHome waits that long before shutting down humidifier). Importantly, the tolerance window can be used to set the timeout set in SafeHome's failure detector protocol.

4 Important Challenges in SafeHome

Concurrency Control–Pessimistic vs. Optimistic: Concurrency control techniques fall along a spectrum. Near the pessimistic end are techniques that: i) disallow routines which touch overlapping devices, ii) disallow routines which touch overlapping safety clauses, iii) have routines "lock" devices (at the edge) before sending commands. Near the optimistic end are approaches such as "last-writer-wins" for device commands, with serializability checks at routine completion. This spectrum is a user convenience tradeoff between responsiveness vs. aborts.We plan to start with pessimistic approaches, and progressively relax them while keeping aborts low. It is an open question which side of this spectrum is the best matched for tomorrow's smart home workloads.

Atomicity, Aborts, and Undoing: If a routine needs to be aborted (e.g., a command failed), naively aborting by sequentially reversing its executed operations may cause a disruptive human experience, e.g., switching on and off lights repeatedly. The problem is worsened by multiple routines aborting. This challenge is unique to smart homes–in the database transaction world intermediate states during an undo are invisible to users. When a set of routines rolls back in SafeHome, no device status must be changed more than once. This requires mechanisms for *consolidated aborting*, wherein groups of routines are aborted together, and their net result is effected on each device at most once. While aborting routines, we can only undo the *states* of affected devices (e.g., set washing machine to off), but actions are impossible to undo (e.g., cannot undo the elapsed wash).

We also note that SafeHome's Atomicity+Durability only requires commands to complete successfully when executed—this is easier to tackle than databases wherein object values are saved at commit time. If the user desires device states to be set a particular when the routine's ends, she can program an explicit check/assert command inside the routine.

Conflicting Safety Properties: Conflicting safety properties need to be either disallowed (if a conflict is likely) or allowed but monitored (if conflict plausible). Responses may include aborting the routine, or ignoring optional commands. Priorities among safety properties are a possible approach, but may be be prohibitive if low complexity is a goal.

Scoping of Safety Properties: To overcome the inflexibility offered by the global set of safety properties, it may be tenable to allow routines or commands to specify locally-scoped safety properties. A locally-scope safety property overrides conflicting global safety properties for the local scope of that routine or command. It is an open question as to what level of density and scale is needed in smart homes, for locally-scoped safety properties to become inevitable.

Long-Running Commands and Routines: Some commands take non-negligible time to execute (e.g., raise shades). These can be captured by two possible approaches: allowing such long-running commands to send the edge a second ack on completion, or allowing devices to expose their intermedi-

ate states [54]. Long-running routines are defined as routines containing such long-running commands. Such routines need to be handled carefully especially w.r.t. conflicts. Aborting a long routine because of a short routine is wasted work, and should be avoided where possible.

Signals in Routines–Exceptions and Interrupts: The routines of a smart home need to be responsive to changing circumstances and changing intentions of the users. If a user (or device) wishes to stop an executing routine, the routine receives an *exception*. If a routine needs to be paused temporarily, it receives an *interrupt*. The code for routines thus needs to come associated with handlers for such exceptions and interrupts. These improve correctness but increase programmer effort–however, it is possible that "template" signal handlers may suffice for most devices or signals.

Goto-Safe States and Dilemma: For a device that fails while a routine is executing a command on it, the edge may not know what state the device restarts in. We address this by having devices restarted in a pre-determined "goto-state". Goto-states are convenient but could cause "Goto dilemmas". If a garage door opener's goto-state is OPEN, burglars may be let in; if it is CLOSED, it might close on a car underneath it. Both are safety violations. How to handle these is an open question. Note that such dilemmas also occur in other cyber-physical environments like self-driving cars [47].

5 SafeHome's Architecture

The architecture of the SafeHome system we are implementing is shown in Figure 2. SafeHome employs a modular design where each component transparently can be swapped with an equivalent mechanism. The key components are:

- **Routine Manager for *Atomicity***: Manages the lifecycle of individual Routines.
- **Safety Checker for *Safety***: Checks for safety property violations. Includes both a static checker and dynamic checker, respectively for compile time and runtime.
- **Concurrency Controller for *Isolation***: Handles concurrency conflicts across routines.
- **Command Deployer for *Durability***: Executes the commands on the devices in a durable manner.
- **State & Health Tracker:** Tracks the current state, health and failures of devices. Notifies Routine Manager and Safety Checker accordingly.
- **Definition Bank:** Contains all routines and safety properties admitted to the system (after static checking).

An "Edge-First" approach: SafeHome is designed with an edge-first approach. On the one hand, cloud dependency hinders scalability, latency, and bandwidth [53] (Section 2). On the other hand, the diversity of smart devices and vendor platforms implies that it is nearly impossible for SafeHome logic itself to run inside all the IoT devices themselves.

The edge-first design cuts through the middle, wherein most of the logic runs on edge devices co-located with the home (e.g, home controllers), with the cloud serving as a backup. Using the edge makes SafeHome generalizable, scalable, efficient, and fast enough to provide SafeHome-ASID properties. Also, it breaks the strong dependency to the cloud, ensuring safety "at-all-times", even in presence of disconnections. For further fault-tolerance we are exploring approaches including a Zookeeper [23] cluster of edges, and (where possible) failover logic in the devices themselves.

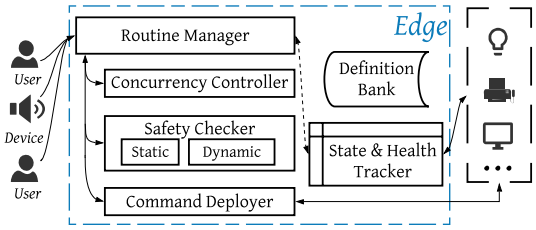

Figure 2: **Architecture of SafeHome's system.**

6 Related Work

Smart home abstractions: Ease-of-use is a well-studied topic for smart homes with various abstractions to simplify automation and management of smart devices [8, 14, 24, 32, 34, 50, 52]. For example, IFTTT [24] represents the home as a set of simple conditional statements, while HomeOS [14] provides a PC-like abstraction for the home where devices are analogous to peripherals in a PC. Beam [39] optimizes the resource utilization by partitioning the applications across devices. However, none of these fully address all challenges on of failures, concurrent commands or miss programming.

Concurrency Control: Concurrency control is a decades-long studied area in database [9]. In the context of IoT systems such as [14, 28, 33, 36] have explored solutions tailored for this cyber-physical environment. While SafeHome utilizes a simple serialization, any of these approaches can replace the modular concurrency controller of SafeHome.

Verification and Type Checking: Detecting conflicts in policies (safety properties in our case) is a long-studied area of research used in several contexts. For example, for network verification, several systems such as NICE [11], Anteater [29], VeriFlow [26] and others [21, 22, 42] detect violating rules and configurations. Similarly, in the IoT space systems such as APEX [54] and [7, 28, 33] enable user-specified conditions and dependencies and verify them. Despite the similarities in these approaches and SafeHome's safety properties, SafeHome is solving a more generic problem (including failures, conflicts and safety violations) where these approaches can be applied to SafeHome's safety checker engine.

7 Conclusion

Designing a safe and reliable home requires carefully designing routine management, declarative safety, concurrency control, and beyond. Many exciting challenges in this direction arise from subtle differences between systems we know (e.g., transactional databases) and systems we are just getting to know (smart homes).

8 Discussion Topics

Expected Feedback, and Discussion Points:

1. What are the downsides of the transaction-based (routine-based) approach for smart homes?
2. Are there examples of complex safety properties that cannot be specified by our grammar?
3. Can the edge fully remove the cloud dependency?
4. Do other ideas from transactional database literature apply (e.g., timestamp ordering, MVCC, etc.)? How about other areas, such as compilers, literature apply?
5. Does SafeHome's idea generalize beyond smart homes, e.g., smart buildings, smart cities, smart factories?

Controversial Points, and Open Issues:

1. The idea of bringing transactional properties into the smart home routines is a radical change from the way smart homes are managed today. The system needs to be designed carefully so that the resulting human-perceived behavior of the system (especially under exceptional circumstances like cascading aborts) is more preferable to the status quo.
2. Safety properties need to be specifiable easily, especially by the lay user. A good UI may be needed for this.
3. While our approach assumes logic on the edge and very little functionality from devices (beyond responding to commands), smarter devices may be able to help with the system's responsiveness and failover. At the same time, the sheer diversity and incompatibility of vendor offerings may be an obstacle to the latter approach.
4. There is an adoptability challenge, especially with closed source assistants, such as Alexa and Google Home. SafeHome can interact with these assistants through an IFTTT binding. However, the definition and invocation of commands/routines should go through SafeHome. This opens a challenge on users adopting SafeHome.

When Does the Idea Fall Apart:

1. If the system is slow to respond, especially to the user. E.g., Pessimistic concurrency control may delay starting of user-initiated routines.
2. If conflicting safety properties and concurrency create chances of deadlocks, which could be especially dangerous if the user is not present to arbitrate such (rare) occurrences.
3. If users are unable to specify safety properties, or there are insufficient safety properties, or routines devolve into single commands, or there is insufficient concurrency. (We think these are unlikely, given the use cases we already see.)

References

[1] Amazon Alexa outage. https://downdetector.com/status/amazon-alexa/news/235561-problems-at-alexa. Last accessed February 2019.

[2] Amazon Alexa. https://developer.amazon.com/alexa. Last accessed February 2019.

[3] Amazon Alexa + SmartThings routines and scenes. https://support.smartthings.com/hc/en-us/articles/210204906-Amazon-Alexa-SmartThings-Routines-and-Scenes. Last accessed February 2019.

[4] Amazon Alexa suffers Christmas outage in Europe. https://www.engadget.com/2018/12/26/amazon-alexa-christmas-outage/. Last accessed February 2019.

[5] Apple Siri. https://www.apple.com/siri/. Last accessed February 2019.

[6] Masoud Saeida Ardekani, Rayman Preet Singh, Nitin Agrawal, Douglas B. Terry, and Riza O. Suminto. Rivulet: A fault-tolerant platform for smart-home applications. In *Proceedings of the 18th ACM/IFIP/USENIX Middleware Conference*, Middleware '17, pages 41–54, New York, NY, USA, 2017. ACM.

[7] I. Armac, M. Kirchhof, and L. Manolescu. Modeling and analysis of functionality in ehome systems: dynamic rule-based conflict detection. In *13th Annual IEEE International Symposium and Workshop on Engineering of Computer-Based Systems (ECBS'06)*, pages 10 pp.–228, March 2006.

[8] Automate.io. https://automate.io/. Last accessed February 2019.

[9] Philip A Bernstein, Vassos Hadzilacos, and Nathan Goodman. *Concurrency Control and Recovery in Database Systems*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 1986.

[10] R. E. Brown, J. D. Brain, and N. Wang. The avian respiratory system: a unique model for studies of respiratory toxicosis and for monitoring air quality. *Environmental health perspectives*, 105(2):188–200, Feb 1997. 9105794.

[11] Marco Canini, Daniele Venzano, Peter Perešíni, Dejan Kostić, and Jennifer Rexford. A NICE way to test openflow applications. In *Presented as part of the 9th USENIX Symposium on Networked Systems Design and Implementation (NSDI 12)*, pages 127–140, 2012.

[12] Comcast outage. https://webdownstatus.com/outages/comcast. Last accessed February 2019.

[13] Abhinandan Das, Indranil Gupta, and Ashish Motivala. Swim: Scalable weakly-consistent infection-style process group membership protocol. In *Proceedings International Conference on Dependable Systems and Networks*, pages 303–312. IEEE, 2002.

[14] Colin Dixon, Ratul Mahajan, Sharad Agarwal, A.J. Brush, Bongshin Lee, Stefan Saroiu, and Paramvir Bahl. An operating system for the home. In *Presented as part of the 9th USENIX Symposium on Networked Systems Design and Implementation (NSDI 12)*, pages 337–352, San Jose, CA, 2012. USENIX.

[15] EE: Average UK Smart Home will have 50 connected devices by 2023. https://www.totaltele.com/500103/EE-Average-UK-Smart-Home-will-have-50-connected-devices-by-2023. Last accessed February 2019.

[16] Google Home. https://store.google.com/us/product/google_home. Last accessed February 2019.

[17] Google Home outage. https://downdetector.com/status/google-home. Last accessed February 2019.

[18] SmartThings outage. https://downdetector.com/status/smartthings/news/224625-problems-at-smartthings. Last accessed February 2019.

[19] Theo Haerder and Andreas Reuter. Principles of transaction-oriented database recovery. *ACM Comput. Surv.*, 15(4):287–317, December 1983.

[20] 'Hey, Google, what's wrong?' Google home 'glitch' gets a fix. https://www.digitaltrends.com/home/google-home-outage-problems/. Last accessed February 2019.

[21] Hongxin Hu, Gail-Joon Ahn, Wonkyu Han, and Ziming Zhao. Towards a reliable *SDN* firewall. In *Presented as part of the Open Networking Summit 2014 ({ONS} 2014)*, 2014.

[22] Hongxin Hu, Wonkyu Han, Gail-Joon Ahn, and Ziming Zhao. Flowguard: building robust firewalls for software-defined networks. In *Proceedings of the third workshop on Hot topics in software defined networking*, pages 97–102. ACM, 2014.

[23] Patrick Hunt, Mahadev Konar, Flavio P. Junqueira, and Benjamin Reed. ZooKeeper: Wait-free coordination for internet-scale systems. In *Proceedings of the 2010 USENIX Conference on USENIX Annual Technical Conference*, USENIX ATC '10, pages 11–11, Berkeley, CA, USA, 2010. USENIX Association.

[24] Ifttt. https://ifttt.com/. Last accessed February 2019.

[25] Internet of Shit. https://twitter.com/internetofshit. Last accessed February 2019.

[26] Ahmed Khurshid, Xuan Zou, Wenxuan Zhou, Matthew Caesar, and P Brighten Godfrey. Veriflow: Verifying network-wide invariants in real time. In *Presented as part of the 10th USENIX Symposium on Networked Systems Design and Implementation (NSDI 13)*, pages 15–27, 2013.

[27] Palanivel A. Kodeswaran, Ravi Kokku, Sayandeep Sen, and Mudhakar Srivatsa. Idea: A system for efficient failure management in smart iot environments. In *Proceedings of the 14th Annual International Conference on Mobile Systems, Applications, and Services*, MobiSys '16, pages 43–56, New York, NY, USA, 2016. ACM.

[28] Chieh-Jan Mike Liang, Börje F. Karlsson, Nicholas D. Lane, Feng Zhao, Junbei Zhang, Zheyi Pan, Zhao Li, and Yong Yu. Sift: Building an internet of safe things. In *Proceedings of the 14th International Conference on Information Processing in Sensor Networks*, IPSN '15, pages 298–309, New York, NY, USA, 2015. ACM.

[29] Haohui Mai, Ahmed Khurshid, Rachit Agarwal, Matthew Caesar, P Godfrey, and Samuel Talmadge King. Debugging the data plane with anteater. In *ACM SIGCOMM Computer Communication Review*, pages 290–301. ACM, 2011.

[30] Mapping the Smart-Home Market. https://www.bcg.com/publications/2018/mapping-smart-home-market.aspx. Last accessed February 2019.

[31] Shrirang Mare, Logan Girvin, Franziska Roesner, and Tadayoshi Kohno. Consumer smart homes: Where we are and where we need to go. In *Proceedings of the 20th International Workshop on Mobile Computing Systems and Applications*, HotMobile '19, pages 117–122, New York, NY, USA, 2019. ACM.

[32] Microsoft Flow. https://flow.microsoft.com. Last accessed February 2019.

[33] S. Munir and J. A. Stankovic. Depsys: Dependency aware integration of cyber-physical systems for smart homes. In *2014 ACM/IEEE International Conference on Cyber-Physical Systems (ICCPS)*, pages 127–138, April 2014.

[34] OpenHAB. https://www.openhab.org/. Last accessed February 2019.

[35] Powahome. https://www.powahome.com/. Last accessed February 2019.

[36] Daniel Retkowitz and Sven Kulle. Dependency management in smart homes. In *IFIP International Conference on Distributed Applications and Interoperable Systems*, pages 143–156. Springer, 2009.

[37] Samsung SmartThings. https://www.smartthings.com/. Last accessed February 2019.

[38] Samsung's SmartThings hub is down for North American users (updated). https://www.engadget.com/2018/03/13/samsung-smartthings-hub-down-north-america/. Last accessed February 2019.

[39] Chenguang Shen, Rayman Preet Singh, Amar Phanishayee, Aman Kansal, and Ratul Mahajan. Beam: Ending monolithic applications for connected devices. In *2016 USENIX Annual Technical Conference (USENIX ATC 16)*, pages 143–157, Denver, CO, 2016. USENIX Association.

[40] Smart home. https://www.statista.com/outlook/279/109/smart-home/united-states. Last accessed February 2019.

[41] Smart home market worth $151.4 billion by 2024. https://www.marketsandmarkets.com/PressReleases/global-smart-homes-market.asp. Last accessed February 2019.

[42] Sooel Son, Seungwon Shin, Vinod Yegneswaran, Phillip A Porras, and Guofei Gu. Model checking invariant security properties in openflow. In *ICC*, pages 1974–1979, 2013.

[43] Stop shouting at your smart home so much and set up multi-step routines. https://www.popsci.com/smart-home-routines-apple-google-amazon. Last accessed February 2019.

[44] P. H. Su, C. Shih, J. Y. Hsu, K. Lin, and Y. Wang. Decentralized fault tolerance mechanism for intelligent iot/m2m middleware. In *2014 IEEE World Forum on Internet of Things (WF-IoT)*, pages 45–50, March 2014.

[45] Top ten ways to keep your bird safe from kitchen dangers. https://veterinarypartner.vin.com/default.aspx?pid=19239&id=6048567. Last accessed February 2019.

[46] TP Link HS105. https://www.tp-link.com/us/download/HS105.html. Last accessed February 2019.

[47] Understanding the fatal Tesla accident on Autopilot and the NHTSA probe. https://electrek.co/2016/07/01/understanding-fatal-tesla-accident-autopilot-nhtsa-probe/. Last accessed February 2019.

[48] How does a virtual assistant work? https://gearpatrol.com/2017/07/19/how-does-a-virtual-assistant-work/. Last accessed February 2019.

[49] How Smart Homes work. https://home.howstuffworks.com/smart-home2.htm. Last accessed February 2019.

[50] Workflow. https://workflow.is/. Last accessed February 2019.

[51] Is Xfinity having an outage right now? https://outage.report/us/xfinity. Last accessed February 2019.

[52] Zapier. https://zapier.com/. Last accessed February 2019.

[53] Ben Zhang, Nitesh Mor, John Kolb, Douglas S. Chan, Nikhil Goyal, Ken Lutz, Eric Allman, John Wawrzynek, Edward Lee, and John Kubiatowicz. The cloud is not enough: Saving iot from the cloud. In *Proceedings of the 7th USENIX Conference on Hot Topics in Cloud Computing*, HotCloud'15, pages 21–21, Berkeley, CA, USA, 2015. USENIX Association.

[54] Qian Zhou and Fan Ye. Apex: Automatic precondition execution with isolation and atomicity in internet-of-things. In *2019 IEEE/ACM Third International Conference on Internet-of-Things Design and Implementation (IoTDI)*, April 2019.