

© 2026 Anna Karanika

Human-Centered Control of Smart Spaces

by

Anna Karanika

Dissertation

Submitted in partial fulfillment of the requirements
for the degree of Doctor of Philosophy in Computer Science
in the Graduate College of the
University of Illinois Urbana-Champaign, 2026

Urbana, Illinois

Doctoral Committee:

Professor Indranil Gupta, Chair and Director of Research
Professor Klara Nahrstedt
Assistant Professor Deepak Vasisht
Dr. Iqbal Mohamed, Samsung AI Centre

Abstract

The Internet of Things has instrumented billions of homes, buildings, farms, and factories with smart devices, yet today’s platforms fail to meet smart-space users’ needs. Control logic sits in vendor clouds and hubs that add latency, cost, and single points of failure; fire-and-forget remote procedure calls (RPCs) leave developers and occupants blind on whether actions take effect; and users navigate a control interface spectrum with no structured support. This thesis argues that smart-space control should be designed around the needs of the people who occupy the space, across three planes of the smart-space stack—the *device action*, *smart-space automation*, and *human interaction* planes.

We propose new abstractions and insights at each of these three planes, spanning distributed systems and human-computer interaction. On the *device action plane*, RASC (Request-Ack-Start-Complete) replaces fire-and-forget RPC with an abstraction that exposes the full action lifecycle atop existing platforms, so automations and occupants can observe and program what their devices do; proofs and experiments confirm fast, low-overhead lifecycle detection and large scheduling gains over state-of-the-art baselines. On the *smart-space automation plane*, CoMesh removes the hub bottlenecks, cloud dependency, and outages of today’s centralized control: it is the first fully decentralized, hub-less control plane for large commercial edge IoT meshes, using group-based coordination with zero-message exchange and locality-sensitive hashing to keep control local, scalable, and fault-tolerant. On the *human interaction plane*, we perform a study of real smart-home owners via surveys and interviews, and derive insights showing that control is a fluid, situational process shaped by reliability and cognitive load rather than ecosystem boundaries, motivating how future smart-space operating systems should actually be built. Though independent, the three contributions share one vision: each makes a plane of smart-space control resilient, observable, and built around real user needs.

Acknowledgments

First and foremost, I thank my advisor, Prof. Indranil Gupta—Indy. Over six years he taught me two things I will carry far past this thesis: how to write, and how not to give up. He also pushed me somewhere I would not have gone on my own: he wanted our systems work grounded in real, practical human needs, and held it to that standard until it was. The human interaction plane of this thesis exists because he saw that gap before I did. I could not have asked for a better advisor.

I am grateful to the rest of my committee. Professor Klara Nahrstedt cared that this thesis be its own thing and shaped the story I told. Dr. Iqbal Mohamed’s hypotheticals—what if the space is a factory, what if the robots move—marked nearly every chapter. Professor Deepak Vasisht was unfailingly supportive, and kept pulling me toward the tools and needs people actually have.

I wish to show my gratitude to the excellent mentors and managers whom I met during my internships: Rebecca Isaacs, Yao Yue and Mihir Nanavati at Twitter, and Li Li, Shiwen Zhao, Ming-Chuan Wu and Emad Soroush at Apple.

This thesis is the product of collaborations I was lucky to be part of, with Kai-Siang Wang, Han-Ting Liang, Shalni Sundram, Rui Yang, Xiaojuan Ma, Jiangran Wang, Ali Zaidi, Ti-Chung Cheng, Yi-Shyuan Chiang, Prof. Camille Cobb, and Prof. Karrie Karahalios. Together they taught me to actually listen to what people say about their homes. Karrie mentored me throughout my PhD—more about life than research, and that turned out to be what I needed most. Rui mentored me as I was entering the program; Kai brought a fresh perspective and a contagious kind of motivation; and Ali mentored me into HCI, a field I would have kept circling from a distance without him. I was also lucky to mentor Kai, Ting, Shalni, and Xirui Yao, an exchange student from Zhejiang University; I suspect I learned as much from them as they from me.

I have been fortunate in my labmates in the Distributed Protocols Research Group

(DPRG): Rui, Xiaojuan, Beomyeol, Cong, Maleeha, Chirag, Sarthak, Xinying, Kai, and Ting. They sat through my practice talks and made our lab in the Siebel Center a nice place to work at. I am grateful to also have met Faria, Shegufta, Mainak, and Shadi, who graduated before I joined and shared their wisdom about industry whenever I was navigating options.

Long before any of this, others set me on the path. Georgios Stamoulis and Kostas Kolomvatsos, my undergraduate advisors at the University of Thessaly in Volos, Greece, encouraged me to pursue a PhD. Spyros Lalis, Dimitrios Katsaros, Christos Antonopoulos, and Nikolaos Bellas are the reason I fell in love with systems, and with distributed systems in particular. And Vana Doufexi and George Thanos, who guided the lab sessions where I first learned to program, gave me the foundation for all of it.

Outside of research, I have been carried by my friends. Melina has been my friend since childhood and stayed one through every move; from my undergraduate years, Madalena, Marianna, Eleftheria, and Evangelia have remained close across all the distance since. At Illinois and elsewhere, I am grateful to Chaitanya, Jovan, Antonis, Xinying, Chloe, Yiran, Filippos, Darman, Nikoleta, Gabby, Richa, Simoni, Yi, Monnet, Makis, Yavneeka, Andres, Thymios, Apostolis, Zois, Olnancy, Milind, David, Kostas, Dimitris, Ben, Adil, and Mubashir—and to the rest of the Greek and Latin dancing communities at UIUC—for making a place far from home feel like one.

Above all, my family. My parents, Eleni and Triantafyllos, gave me everything that made this possible, and stood by me through all of it even when they would rather I had chosen something easier, or closer to home. My uncle Yiorgos, my aunts Anthi and Despoina, and my cousins Nikos, Aphroditi, Roula, and Nikos have been a constant. I have no siblings, but I made siblings of choice out of my cousins and closest friends. There has been a great deal of distance between us over these years, and not once has it pushed us apart—if anything, it brought us closer. They supported me wholeheartedly, in a way I could not repay. I am forever grateful.

*To my family—
the one I was given, and the one I chose.*

Contents

List of Abbreviations	9
Chapter 1 Introduction	10
1.1 Background and Motivation	10
1.2 Thesis Contributions	14
1.3 Broader Impact	19
1.4 Thesis Organization	20
Chapter 2 RASC: Enhancing Observability and Programmability in Smart Spaces	21
2.1 Introduction	21
2.2 System Model	26
2.3 Design Goals	28
2.4 Observability in RASC	29
2.5 Building Atop the RASC Abstraction	37
2.6 Implementation	45
2.7 Experimental Evaluation	46
2.8 Discussion	54
2.9 Related Work	56
2.10 Conclusion	57
Chapter 3 CoMesh: A Decentralized Control Plane for Commercial Edge IoT Meshes	58
3.1 Introduction	59
3.2 Challenges and CoMesh’s Overview	64
3.3 System Model	66
3.4 CoMesh’s Mechanism of Coteries	67
3.5 Managing Devices and Routines in CoMesh	73
3.6 Formal Analysis	76
3.7 Experimental Evaluation	81
3.8 Related Work	93
3.9 Summary	94
Chapter 4 Control in Context: How Smart Home Users Navigate Proxy-Based Control Schemes	95
4.1 Introduction	95
4.2 Related Work	100

4.3	Methods	106
4.4	Results	113
4.5	Discussion	125
4.6	Limitations and Future Work	134
4.7	Conclusion	135
Chapter 5	Conclusion and Future Work	137
5.1	Summary of Contributions	137
5.2	Future Research Directions	138
References		143
Appendix A:	Control in Context—Supplemental Material	179
A.1	Main Survey Questions	179
A.2	Interview Script	191
Appendix B:	Other Work	198

List of Abbreviations

API	Application Programming Interface
DAG	Directed Acyclic Graph
GUI	Graphical User Interface
HCI	Human-Computer Interaction
IoT	Internet of Things
LLM	Large Language Model
LOC	Lines of Code
LSH	Locality-Sensitive Hashing
OS	Operating System
P2P	Peer-to-Peer
RASC	Request-Ack-Start-Complete
RPC	Remote Procedure Call
RTT	Round-Trip Time
STA	Sense-Trigger-Actuate

Chapter 1 Introduction

1.1 Background and Motivation

The Internet of Things (IoT) has evolved well beyond its early phase, with around 21 billion connected devices in deployment as of 2025, forecast to reach roughly 40 billion by 2030 [1]. IoT permeates homes, workplaces, farms, Industry 4.0 settings, entertainment venues, and beyond [2], [3], [4]. Smart commercial buildings alone account for around 2 billion connected devices today, projected to reach 4.12 billion by 2030, with the building IoT market growing from roughly \$64 billion in 2024 to over \$100 billion by 2030 [5]. As deployments scale, they increasingly span *mesh networks* of heterogeneous devices—smart devices capable of compute and storage alongside simpler devices that only receive commands—creating densely populated edge computing environments across both residential and commercial settings; Figure 1.1 illustrates three such commercial settings—a smart building, a smart farm, and a smart factory.

Table 1.1: **Taxonomy of smart spaces addressed in this thesis and the control challenges each presents.**

Smart Space	Key Control Challenges	This Thesis
Residential home (10s of devices)	Navigating the control spectrum; debugging silent automation failures	Chapter 4, Chapter 2
Commercial building (100s–1000s)	Hub overload; cloud dependency; control scalability	Chapter 3, Chapter 2
Smart farm (distributed, outdoors)	Connectivity gaps; unattended local operation	Chapter 3, Chapter 2
Smart factory (industrial)	Strict data-locality requirements; operational continuity	Chapter 3, Chapter 2

We use the term *smart space* to refer to any physical environment instrumented with IoT devices for automated sensing and actuation. Smart spaces span a wide range of scales and deployment contexts—from a residential home with a few dozen consumer devices to a

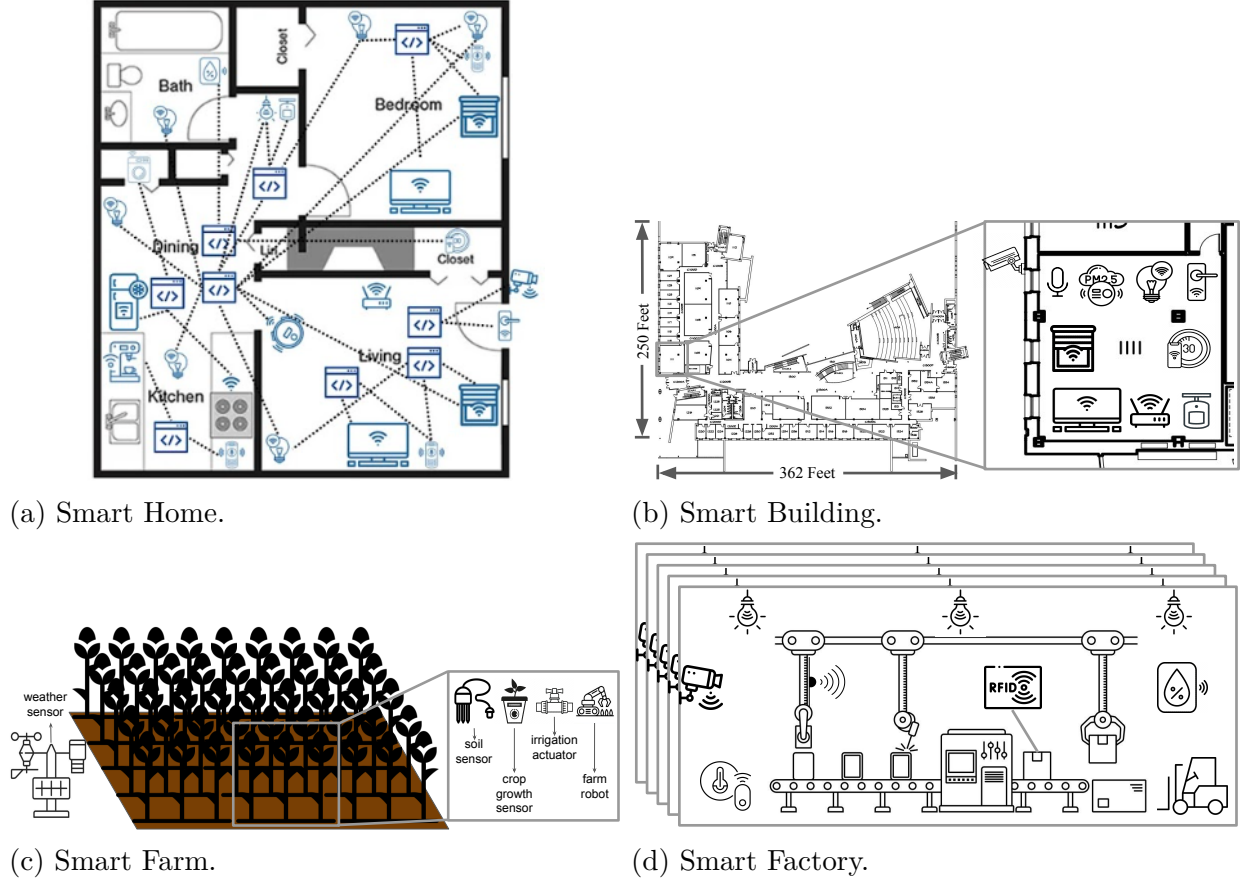


Figure 1.1: **Four Edge Settings:** (a) **Smart Home** [1], (b) **Smart Building** [6], (c) **Smart Farm** [7], (d) **Smart Factory** [4]. *The Smart Building map from (b) and Smart Farm from (c) drive the experiments in the CoMesh chapter (Chapter 3).*

commercial smart building with hundreds or thousands of sensors and actuators, to smart farms and industrial factories with large physical footprints. Table 1.1 provides a taxonomy of the smart space types addressed in this thesis.

Despite their differences in scale and purpose, all smart spaces share a common computational substrate: the *Sense-Trigger-Actuate* (STA) paradigm, in which devices sense their environment, automation rules evaluate conditions and fire, and commands actuate devices. These workloads—packaged as automations, routines, or scripts—are how occupants and operators program their environments to respond to the world around them.

Modern IoT stacks must be evaluated through two complementary lenses.

System-facing metrics—latency, reliability, fault tolerance, privacy, and cost-efficiency—ensure that automations trigger promptly, survive device or network

failures, and protect sensitive telemetry. **User-facing metrics**—fine-grained control, observability, debuggability, and overall usability—determine whether occupants can understand and shape the behavior of their increasingly complex environments. Together, these metrics define the standard against which any scalable, human-centered IoT platform must be evaluated. This thesis contributes to both dimensions, combining systems design and implementation with empirical human-computer interaction (HCI) research.

Today’s dominant IoT architecture routes every sensor reading and actuation request through a vendor cloud backend, which creates three intertwined pain points. First, **privacy risks** arise because raw telemetry must transit and persist in third-party infrastructure; keeping control logic on-premise eliminates this exposure entirely [8], [9]. In high-security industrial settings—semiconductor fabs and pharmaceutical facilities, for instance—operators routinely prohibit any data from leaving the premises to protect proprietary processes, a constraint cloud-centric platforms cannot satisfy by design. Second, **reliability degrades** when every automation depends on an Internet path, particularly during network or cloud service outages [10]. Third, **cost scales unfavorably**: servicing 500 devices that ping once per second can incur fees approaching \$2,500 per month on mainstream IoT cloud tiers [11], while local control removes usage-based bills altogether. Together, these pressures make edge-native, on-premise control an attractive foundation for smart environments that must be private, dependable, and economically sustainable. Crucially, the hardware needed to realize this vision is already in place: smart devices in both residential and commercial settings sit idle for the vast majority of their operational lifetime [12], offering a ready substrate for distributed control logic without requiring any additional infrastructure investment.

Beyond infrastructure placement, the programmability of smart spaces remains a fundamental challenge. IoT automations—routines that chain sensor readings to device actuations—are central to how occupants live with their environments. Yet the tools available to both users and developers today are impoverished in two ways. First, users

navigate a rich spectrum of control options, from physical device interfaces to centralized proxy-based hubs, with limited system support for moving between them fluidly. Platforms today are designed around vendor ecosystems and commercial integration goals, not around the actual, context-dependent ways people rely on their devices. Second, the control primitive underlying virtually all automation platforms—the Remote Procedure Call (RPC)—gives no feedback on whether a physical action has actually taken effect [13]. This leaves both users and developers without the observability needed to build reliable, debuggable workflows. Closing these gaps is the central challenge this thesis addresses.

A note on large language models (LLMs) is warranted, given their rapid rise over the course of this research. LLMs largely do not alter the conclusions of this thesis, but they can help realize its vision. The device action plane’s need for an observable control abstraction is orthogonal to how automations are authored: RASC’s conclusions stand unchanged, though LLMs could ease its implementation—for instance, by helping developers and users specify causal dependencies among actions. At the human interaction plane, Control in Context’s findings characterize human control practices; LLMs, vision, and other machine learning advances do not change these fundamentals, though we note that had LLMs been as prevalent at the time of our study as they are today, participants might already have been using them to program routines or troubleshoot—making LLM-mediated interaction a new point on the control spectrum worth studying—and such technologies can help realize the design implications we derive. Finally, LLMs may make programming routines easier, but they do not change the smart-space automation plane’s case for decentralized coordination: CoMesh’s design is agnostic to how the routines it executes were written. The concluding chapter discusses LLM-related research directions this thesis opens.

Thesis Statement. This thesis shows that it is possible to build networks and systems for smart spaces that are attuned to the needs of the people who inhabit them, advancing this vision across three *independent* planes of the smart-space stack. At the *device action plane*,

RASC introduces a control abstraction that brings action-level observability and programmability to IoT automations. At the *smart-space automation plane*, CoMesh provides a decentralized infrastructure that removes hub bottlenecks and cloud dependence. Finally, at the *human interaction plane*, Control in Context contributes an empirical account of how and why real occupants navigate the control spectrum.

1.2 Thesis Contributions

Table 1.2: **Contributions of this thesis, their objectives, and evaluation methodology.**

Work	Plane	Objective	Evaluation	Venue
RASC [14]	Device action	Observability & programmability abstraction for IoT device actions	System measurements	USENIX NSDI 2026
CoMesh [15]	Smart-space automation	Scalable & fault-tolerant abstraction for commercial edge IoT mesh automations	System measurements	IEEE TNSM 2026
Control in Context [16]	Human interaction	Smart home control-interface navigation framework	HCI analysis	ACM CHI 2026

This thesis makes three concrete contributions to the control of smart spaces—one at each plane of the smart-space stack, unified by a single user-aware vision (Figure 1.2). It combines systems design and implementation with empirical user research. Table 1.2 summarizes the three contributions and their evaluation methodology.

- **The device action plane: observable, programmable control (Chapter 2).**

How do automations and occupants observe and program actions? The RPC primitive underlying all IoT platforms is fire-and-forget, leaving developers and occupants without feedback on whether actions have actually taken effect. RASC [14] is a new control abstraction that exposes the full action lifecycle, so that automations—and the people who write them—can observe whether an action started, completed, or failed, and program causal dependencies among actions, all without requiring any device modifications.

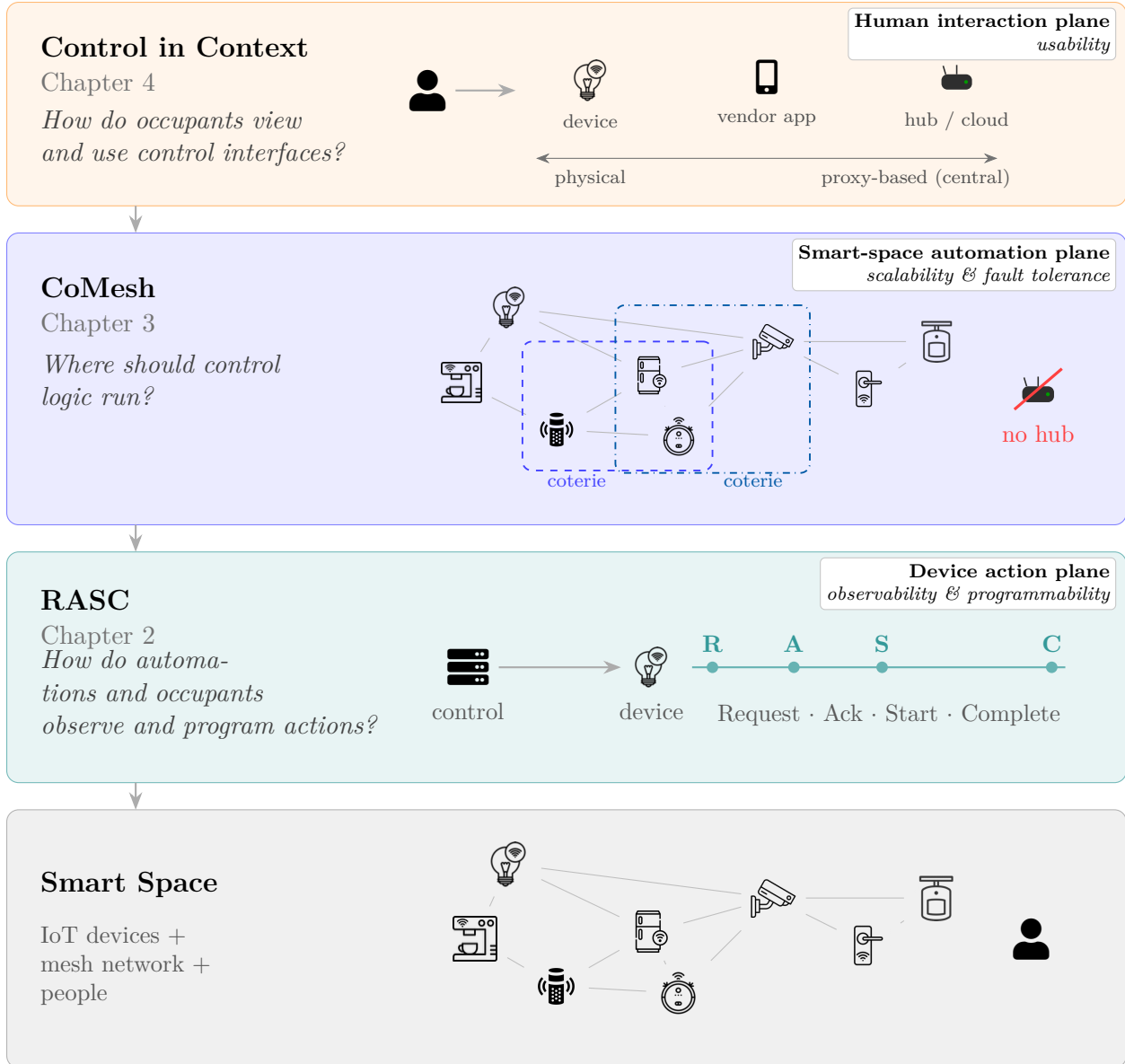


Figure 1.2: **Three user-aware planes of smart-space control, designed around the needs of the people who inhabit the space.** The device action plane (*RASC*) exposes each action’s Request-Ack-Start-Complete lifecycle; the smart-space automation plane (*CoMesh*) decentralizes where control logic runs; and the human interaction plane (*Control in Context*) characterizes how occupants view and use control interfaces. All three operate over a shared smart space of devices, mesh network, and people.

- **The smart-space automation plane: scalable, fault-tolerant control (Chapter 3).** *Where should control logic run?* Today’s platforms concentrate control logic in a central hub or cloud, creating bottlenecks, single points of failure, and cloud dependency. CoMesh [15] is the first fully decentralized, hub-less control

plane for large commercial edge IoT meshes: it spreads control across the devices themselves, so the system *scales* to thousands of devices and *tolerates* device failures, churn, and mobility without any central point of failure.

- **The human interaction plane: usable control interfaces (Chapter 4).** *How do occupants view and use control interfaces?* Smart-home users interact through a spectrum of command interfaces—from pressing a physical button on a device, to a per-brand mobile app, to a centralized hub application such as Google Home or Alexa that routes all commands through a cloud backend. These represent different interface architectures, each with different latency, reliability, and failure-mode characteristics. System designers have long assumed centralized hub interfaces are users’ primary mode, yet real behavior is far more nuanced. Control in Context [16] is the first empirical characterization of how and why users move across this interface spectrum in practice, grounding future interface and system design in observed behavior rather than assumptions.

These are independent contributions unified by a single vision—user-aware design—rather than a pipeline: no insight “flows” from one to the next, and none subsumes another. Together they make three planes of the smart-space stack—the *device action plane* (how actions are observed and programmed), the *smart-space automation plane* (where control logic runs), and the *human interaction plane* (how occupants view and use control interfaces)—resilient, observable, and built around the people who inhabit the space. Figure 1.2 situates each contribution accordingly.

1.2.1 RASC: Enhancing Observability and Programmability in Smart Spaces

IoT platforms today rely on RPC, a fire-and-forget primitive that provides no feedback on whether a physical action has started, completed, or failed. This leaves both users and developers without the observability needed to build reliable routines, reason about device

state, or diagnose failures. We introduce RASC [14] (Request-Ack-Start-Complete), a new abstraction for IoT device control that exposes the full lifecycle of every action, and Rascal, its implementation atop existing RPC-based IoT platforms.

RASC’s contributions are:

- The RASC abstraction, which extends RPC with four lifecycle states—Request, Acknowledgment, Start, and Complete—enabling both users and higher-level automation systems to reason about action progress in real time.
- An adaptive polling strategy that detects when actions start, complete, or fail with minimal overhead, without requiring any device-side modifications.
- Scheduling policies over a directed acyclic graph (DAG) of actions that express causal dependencies, enabling routines that react dynamically to action outcomes rather than relying on fixed time windows.
- Integration with Home Assistant via Rascal, a widely used open-source smart-home platform, and trace-driven evaluation showing that Rascal detects action completion within 2–16 seconds in over 90% of cases and that its scheduling policies outperform state-of-the-art baselines by 10–55%.

1.2.2 CoMesh: A Decentralized Control Plane for Commercial Edge IoT Meshes

Today’s smart spaces route all STA coordination through a central hub or cloud—a single point of failure that drives outages, exposes private telemetry, and incurs prohibitive cloud costs, and that does not scale to the hundreds-to-thousands of devices in commercial edge deployments such as smart buildings, farms, and factories. We present CoMesh [15], the first fully decentralized, hub-less control plane for STA operations over large commercial edge IoT meshes. CoMesh organizes smart devices into overlapping subgroups called *coteries* that collectively monitor devices and manage routines using zero-message exchange

protocols, quorum-based agreement, epoch-based migration, and locality-sensitive hashing (LSH). The result is a self-healing architecture that adapts to device failures, churn, and mobility without human intervention.

CoMesh’s contributions are:

- Design and implementation of CoMesh using a novel combination of distributed systems techniques: load-balanced STA operations over space and time; zero-message member selection; quorum-based agreement; and LSH-based locality to minimize inter-device communication.
- Formal proofs of CoMesh’s safety, liveness, and progress properties.
- Experimental evaluation on Raspberry Pi deployments and large-scale simulations using real building maps and automation workloads, demonstrating scalability, fault tolerance, and low latency.

1.2.3 Control in Context: How Smart Home Users Navigate Proxy-Based Control Schemes

Smart home platforms are designed around vendor ecosystems and commercial integration goals, not the actual, context-dependent ways people live with and rely on their devices. Prior work examined individual control schemes in isolation or proposed system architectures without grounding them in observed user behavior. We present Control in Context [16], a mixed-methods study characterizing how and why smart home users navigate the spectrum from physical device interaction to proxy-based centralized control across everyday tasks, device failures, and evolving home setups.

Control in Context’s contributions are:

- A survey of $N = 60$ smart-home device owners and semi-structured follow-up interviews with $N = 8$ participants, revealing that control is a fluid, reflective process shaped by tradeoffs between convenience, reliability, and cognitive load.

- Evidence that users move freely along the control spectrum guided by situational factors such as task type, perceived reliability, and device location—not by abstract ecosystem boundaries.
- Characterization of how troubleshooting workflows interact with control scheme preferences, showing that failures lead users to reassess strategies and sometimes abandon devices or platforms entirely.
- Design implications for future smart-home systems that support movement across the control spectrum and treat the user—not the device ecosystem—as the center of design.

1.3 Broader Impact

This thesis advances a local-first, human-centered vision for IoT, with benefits for commercial ecosystems, everyday occupants, and the research community:

- **Progress-aware automation (RASC):**
 - Multi-vendor automation tooling without brittle polling or closed software development kits.
 - Fewer silent failures: occupants can inspect, pause, or roll back automations like debugged programs.
 - A common API for reasoning about—and formally verifying—workflow correctness.
- **Decentralized edge control (CoMesh):**
 - Runs on the idle devices already present [12]—no recurring cloud fees, less wide-area network traffic, and no single-hub failure point.
 - Keeps sensitive telemetry on-premise, easing regulatory exposure.

- Keeps critical automations (locks, alarms) working through network and cloud outages.
 - Seeds sustainable, low-carbon IoT research—constrained, mostly-idle devices can shoulder control once reserved for data centers.
- **Human-centered control insights (Control in Context):**
 - Ground-truth evidence on where occupants struggle cuts device returns and support volume.
 - Motivates transparency-driven designs that make dense smart spaces accessible and trustworthy for less-technical users.
 - Clearer mental models and faster diagnostics for occupants in dense smart spaces.

Collectively, this thesis turns once vendor-locked properties—privacy, latency, and programmability—into parameters that occupants and developers can tune to fit their priorities—choosing how intelligent their space is and where that intelligence lives.

1.4 Thesis Organization

The remainder of this dissertation is organized as follows. Chapter 2 presents Rascal, our implementation of the RASC abstraction, including case studies, dynamic scheduling algorithms, and a trace-driven evaluation. Chapter 3 presents CoMesh, our decentralized control plane for commercial edge IoT meshes, including formal analysis and evaluation on Raspberry Pi deployments and large-scale simulations. Chapter 4 presents Control in Context, our mixed-methods user study characterizing how and why smart home users navigate the spectrum from physical to proxy-based control. The dissertation concludes with a summary of contributions and a discussion of future directions.

Chapter 2 RASC: Enhancing Observability and Programmability in Smart Spaces

This chapter addresses the *device action plane* of smart-space control (Section 1.1)—*how do automations and occupants observe and program actions?*—and applies across all deployment types in Table 1.1. We present **RASC** [14] (Request-Ack-Start-Complete) and its implementation, Rascal, a new abstraction and system for IoT device control that addresses a fundamental gap in today’s smart home platforms: the inability to know whether a physical action has actually taken effect. IoT platforms today rely on RPC, a fire-and-forget primitive that gives no feedback on action progress or failure—leaving both users and smart home developers without the observability needed to build reliable routines, reason about device state, or diagnose failures. Rascal exposes the full lifecycle of every action and introduces DAG-aware scheduling that enables automations to express causal dependencies, bringing observability and programmability to IoT settings.

This chapter is organized as follows. Section 2.1 introduces the problem and presents motivating examples. Section 2.2 defines the system model. Section 2.3 states our design goals. Section 2.4 describes the RASC abstraction and our adaptive polling strategy for action progress detection. Section 2.5 presents the programmability layer built atop RASC, including causality abstractions, dynamic scheduling, and our DAG-TL scheduler and STF and RV reschedulers. Section 2.6 describes our Home Assistant integration. Section 2.7 reports our experimental evaluation. Section 2.8 discusses design considerations. Section 2.9 surveys related work and Section 2.10 concludes.

2.1 Introduction

In the last decade, industry and users have moved away from managing individual IoT (Internet of Things) devices to programming and managing IoT device *collections*. IoT permeates our homes [2], workplaces [17], farms [3], Industry 4.0 [4], entertainment venues [18], etc. Smart buildings contain 2 B devices, expected to reach 2.5 B devices and

a \$90 B market by 2027, and 4.12 B devices by 2030 [5], [6]. In a single deployment, 10s to 100s of IoT devices may be programmed and managed via automations [19] containing a combination of actions [20], [21], routines [22], [23], IFTTT (If This Then That) [22], [24], scripts, etc. An *action* is a single command sent to a single device, from a home hub (e.g., Alexa, Google Home hub) or a user device (e.g., phone, tablet, etc.). For instance, the Google Home API [20] contains over 50 defined actions, e.g., `blinds.OpenClose`, `SetFanSpeedRelative`, `Cook`, `Dispense`, etc. The most common program written by users is a *routine*: a sequence of several actions, which can be triggered conditionally by time, sensors, or manually by the user.

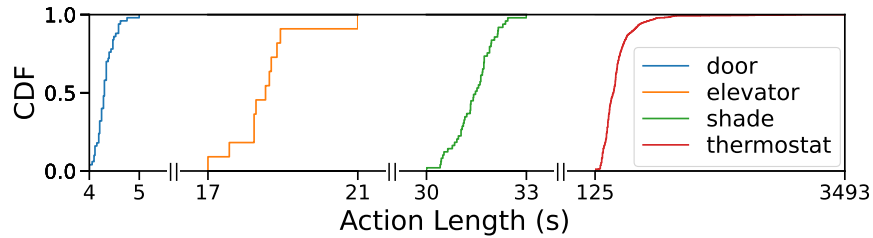


Figure 2.1: **IoT actions vary in length.** *Actions: door: close, elevator: up 1 floor, shade: open, thermostat: heat $68.x$ to $69.y$ °F.*

RPCs and IoT Actions: Today, IoT actions predominantly rely on the traditional RPC (Remote Procedure Call) abstraction [13]. RPCs are simple, widely understood, and accepted, and have many stock implementations [25], [26], [27], [28]. An RPC consists of a single *Request* from a hub or personal device towards the IoT device (or its proxy web service), and a single *Reply* in the reverse direction.

However, for IoT settings, it is challenging to map the Reply to the appropriate point in the action’s execution. This is because in user-facing settings like smart spaces, many actions are non-instantaneous, taking seconds or even minutes to execute. Physical devices like windows and doors, shades and awnings, kitchen appliances, printers, etc., all take several seconds to execute an action (e.g., Google Home API’s actions `blinds.OpenClose`, `SetFanSpeedRelative`). That is, many actions in IoT settings are **long actions**. Devices like sprinklers, ovens, HVAC systems (heating, ventilation, and air conditioning), etc., may

take minutes or hours (e.g., Google Home API’s actions **Cook**, **Dispense**, etc.). Even “fast” devices like locks, light bulbs, fans take a fraction of a second to finish an action. Together, these create a non-trivial time gap between starting and completing an action. Figure 2.1 shows our measurements from devices in our office building.

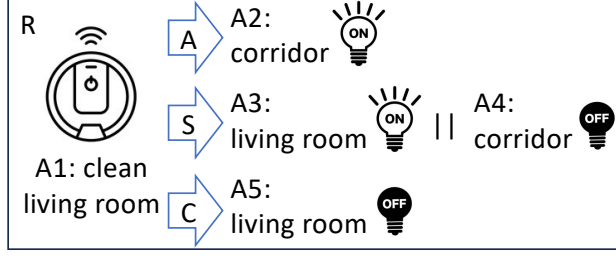
Further, there is also a gap between the command’s reception and start, due to **preconditions** for starting a command (e.g., coffee maker’s reservoir must have water). Finally, the **action may fail** (e.g., main door lock is suddenly jammed) and create safety-violating situations.

Returning to RPCs, this diversity in the nature and length of actions in IoT spaces raises the question of *which* version of RPC to use. Specifically—*when* is the Reply sent back? Today’s RPC implementations are *forced to choose* mapping of the Reply to one of three choices—either: (1) *Acknowledgment*: sent back by the device or its cloud service immediately when it receives the action (we call this *RPC-A*), but before the device has started executing the action, *or* (2) *Start* of the action: sent back by the device or its service when the action starts (we call this *RPC-S*), *or* (3) *Completion* of the action: sent back by the device or its service when the action is completed by the IoT device (we call this *RPC-C*). Many commercial smart home deployments today use RPC-A combined with state updates [29], [30], while RPC-C is commonly employed in robotics settings [31], [32].

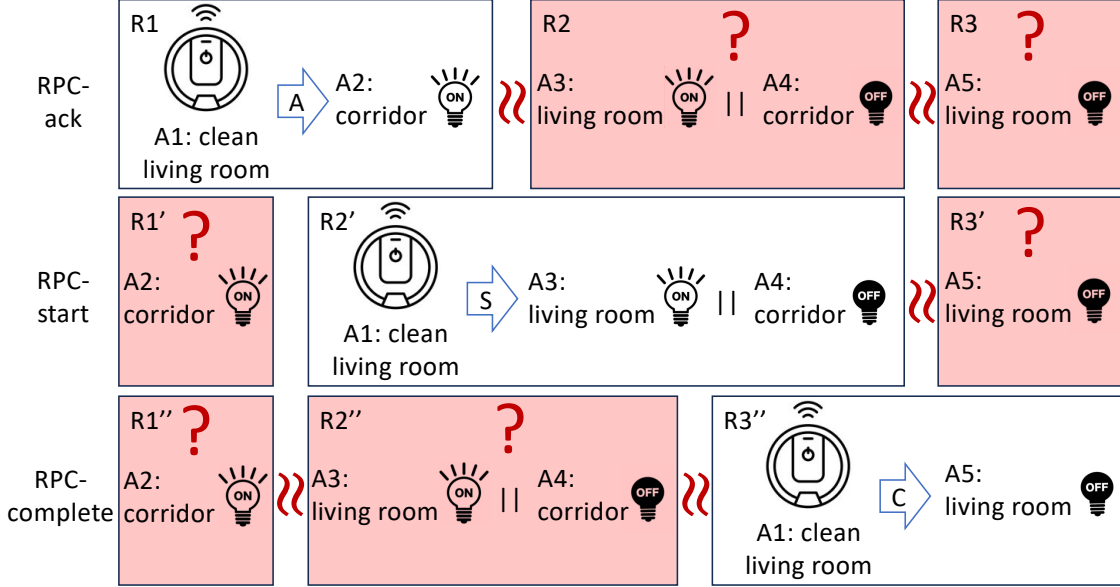
Table 2.1: **Desired Properties.** $\sim$ means non-trivial design needed.

Property	Goal	RPC-A	RPC-S	RPC-C	RASC
<i>Observability</i>	Progress updates (Section 2.4.1)	✗	✗	✗	✓
	Failure detection (Section 2.4.2)	✗	✗	~	✓
<i>Programmability</i>	Diverse dependencies (Section 2.5.1)	✗	✗	✗	✓
	Dynamic scheduling (Section 2.5.2)	✗	✗	~	✓

Goals for an RPC Alternative: The inherent *one-shot* nature of an RPC is a mismatch with the *long-running* nature of IoT actions. We posit that the call-response for IoT device actions must adhere to two inter-related principles: (1) **Observability**: the ability to track a device action; (2) **Programmability**: the ability to write and efficiently



(a) Single routine for energy-saving vacuum atop RASC.



(b) Ineffective RPC-based energy-saving vacuum routines. Pink routines need complex—or unavailable—triggers (?; e.g., “vacuum in living room” $\wedge$ “cleaning/returning”). With suboptimal triggers, cross-routine conflicts ($\llcorner$) arise or actions fire unintentionally.

Figure 2.2: **Routine example with dependencies on *Acknowledgment* (A), *Start* (S) and *Completion* (C) among actions.** *The vacuum requires light to navigate.*

run *expressive* programs.

Observability requires that the hub (or user device) receive **progress updates** at critical action execution points. If any key action part fails, internal mechanisms must **detect failure fast**, and mitigate. Second, programmability implies that programs support expressive and **diverse dependencies** among actions (inside a program, and across programs), e.g., start the next action *A2* earlier, *during* execution of a previous dependent action *A1*, after *A1* has crossed key internal points (rather than waiting for *A1* to finish). These dependencies require us to design **dynamic scheduling** for concurrent smart-space actions. Rescheduling is vital because varying action lengths and tail latency

(Figure 2.1) cause executions to deviate from original estimates and schedules. This intertwined nature of observability and programmability—summarized in Table 2.1’s four goals—parallels challenges distributed systems [33], [34], Kubernetes [35], and SDNs [36].

A New Abstraction: This chapter proposes *RASC* (Request-Ack-Start-Complete), a new abstraction for IoT devices, as an alternative to RPC. RASC does not replace RPC, but instead can be built atop RPC. RASC provides Replies at three critical action lifecycle points: **Ack** (the action is received by the device but not yet started), **Start** (the device starts the action), and **Complete** (the device finishes the action). Naturally, some of these replies may overlap (e.g., if the action is short), but each is essential for longer actions to satisfy Table 2.1’s goals.

Consider a robotic vacuum that requires ambient light in its work area (for the robot’s camera). Figure 2.2a shows a routine (a program with a set of actions) named *R* expressed with diverse action dependency types. Because the vacuum needs light for its camera to navigate, initially the corridor light is switched on so that the vacuum can find its way to the living room (since the precondition for the action is that the vacuum reach the living room). After that, the living room light is kept on for vacuuming, but the corridor light is switched off. When the vacuum has completed and docks in the living room, the living room light is switched off. Similar routines exist involving lawn-mowers [37], tele-presence robots [38], etc.

Figure 2.2b shows that to *correctly* implement *R* via RPC variants, *R* needs to be split into several routines. Each split is problematic because (i) there are no good triggers for the pink routines (e.g., vacuum location cannot be used in trigger clauses), and (ii) if users go for clauses provided by the APIs, which tend to be simple, they will not get the intended result.

Our system Rascal implements the RASC abstraction. Assuring Table 2.1 entails several challenges. First, we have to build a **progress/failure detector** atop RASC, that minimizes detection time while respecting device constraints. For poll-only devices, it must

poll often enough to track progress and catch failures promptly without overwhelming devices. Second, atop RASC when we provide support for expressive routine dependencies, we must innovate **new dynamic scheduling algorithms** that reduce end-to-end latency (from routine arrival to completion). Third, a key principle in our design is backwards compatibility: to make Rascal immediately deployable atop today’s ecosystems, IoT devices and their vendor services cannot be modified. This forces us to work within the constraints of existing RPC interfaces. This is challenging as (i) RASC is more expressive than RPCs, and (ii) RPCs may go either via the IoT cloud service, or directly to the device.

This chapter makes the following contributions:

- We present a new expressive RPC-enhancing abstraction called RASC, for IoT settings.
- We build the Rascal system that implements RASC over existing RPC-based IoT APIs. To support action progress updates and failure detection, we propose new techniques for efficient device polling. We also propose new dynamic scheduling policies for diverse action dependencies.
- We integrate Rascal with the popular and open-source home automation system called Home Assistant [39].
- We measured action execution times for various devices in office buildings. We use these and sets of real routines to perform trace-driven evaluation. We find that (i) Rascal detects completion within 2–13 RPCs and 2s–16s over 90% of the time, and (ii) our routine scheduling policies outperform state-of-the-art by 10%–55%.

2.2 System Model

In today’s deployments, devices communicate in one of five ways [40]: *Cloud Pull*, *Cloud Push*, *Local Pull*, *Local Push*, *Assumed State*. Here, *Cloud* means via the cloud service (device vendor’s cloud service), while *Local* means locally directly from the device via Wifi,

Bluetooth, Zigbee, etc. *Push* means the device updates the cloud service whenever its local state changes, while *Pull* means the cloud service has to poll the device for any state changes. For very old devices that allow neither pull nor push, the cloud service has to assume a state based on the last action. While the push variant naturally leans into providing updates for start and complete (and also intermediate states along the way), it is mostly adopted by sensor devices. Pull variants are widely prevalent—a cloud service or home hub continuously polls the device to detect updates. In fact, during our experiments, we have observed that at least 30% of sampled vendors supported by Home Assistant only offer pull options.

The execution of a single action on an IoT device has three components: (1) a *network component* (transmission of request and response messages among the mobile, hub, cloud, device), (2) a *contextual component* (virtual and physical) comprising preconditions (either physical or safety-related) needed for the device to execute the action, and finally (3) a *physical component* (device executing the action). RASC argues that the three stages are spatially and conceptually distinct. We assume IoT devices satisfy a few properties:

- *IoT devices and their vendor services cannot be modified.*
- *Only one action executes on a device at a time.* This is typical for today’s devices, some of which maintain a queue (e.g., HP printers [41]) while others reject new requests when busy (e.g., Nuki locks [42]).
- *Sufficient polling rate.* To query the state of a pull-based device, we assume arbitrarily frequent pulls are allowed, e.g., Hue bulbs allow polling every second via Hue Bridge [43].
- *Failures/Speed.* IoT devices may be arbitrarily slow or fail to execute actions. The network may delay or drop packets, and IoT devices and their vendor services need not be synchronized. We assume the reliability of devices beyond our purview (cloud services, hub, phones, etc.).

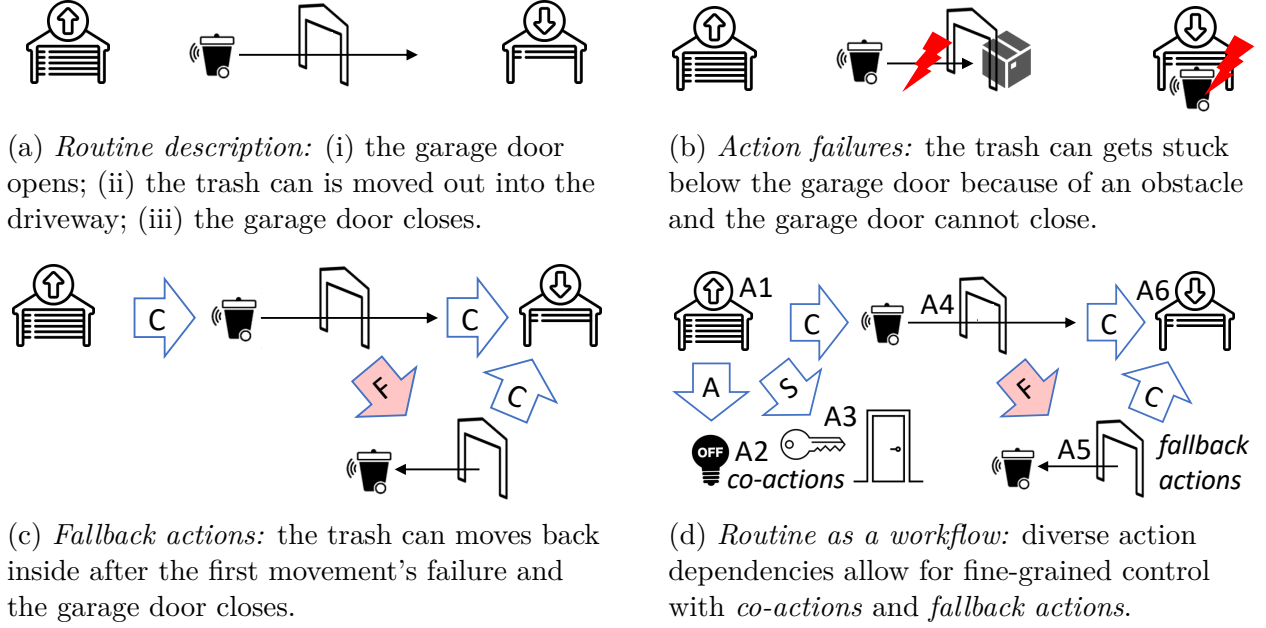


Figure 2.3: **Routine with exception handling.** *Garage door opens, automatic trash can [44] goes out, garage door closes. Letter inside arrow indicates trigger event: S=Start, C=Complete, F=Failure.*

2.3 Design Goals

We discuss Table 2.1’s key goals in detail.

Action progress updates. This goal implies that the primitive should provide feedback at key points of the action execution: acknowledgment, action start, and action end (and if the device supports it, during the action). Doing so may require polling, and this polling needs to be efficient.

Action failure detection. When an action fails, the primitive should detect it quickly. This property may be implemented atop RPC-complete with the use of a timeout after the expected action length is over; however, that is not trivial since there is no fixed length for each action.

Diverse action dependencies. The primitive must support rich inter-action dependencies. For a given example (garage door opens and closes for a trashcan), Figure 2.3(a,b) show two scenarios, and Figure 2.3(c,d) show two variants. In Figure 2.3d, A2 and A3 are *co-actions* of A1. The co-action concept allows (i) the light to turn off

before the garage door starts opening, so that insects are not drawn indoors, and (ii) the inside door to lock only if the garage door starts opening; if *A1* fails, no safety gap occurs. If *A4 fails* (Figure 2.3b), the *fallback action* *A5* returns the trashcan to its original spot, and, finally, *A6* closes the garage door to prevent exposure to robbers. Implementing Figure 2.3d via RPC requires users stitching multiple routines (roughly four here—one extra for every new dependency type) guarded by complex predicates to emulate progress points, to block unintended triggers. This is prohibitive.

Dynamic action scheduling. When action lengths differ from the schedule, the primitive should reschedule subsequent actions. Previous smart home routine scheduling [45] (Figure 2.3a) ignores unbounded action lengths and diverse dependencies (Figures 2.2a and 2.3d), relying instead on static schedules. Action durations are context-dependent (e.g., floor area, dirt level for vacuum, etc.), and failures are unpredictable at compile or schedule time. The schedule must adapt to ensure: (a) *safety*, i.e., no actions trigger concurrently for the same device (which might cause action rejections), (b) upheld action dependencies, and (c) early starts for actions if predecessors finish ahead of schedule. RPC-A and RPC-S cannot handle this, and RPC-C only detects completion. RASC is more powerful.

2.4 Observability in RASC

The goals of action progress updates and failure detection (Table 2.1) can be served via a mechanism that *periodically polls* the device for its current state. This polling needs to (i) satisfy user-specified tolerances for detecting failures, (ii) balance detection latency vs. device/network load, and (iii) adapt to evolving action time distributions (i.e., wrong estimates).

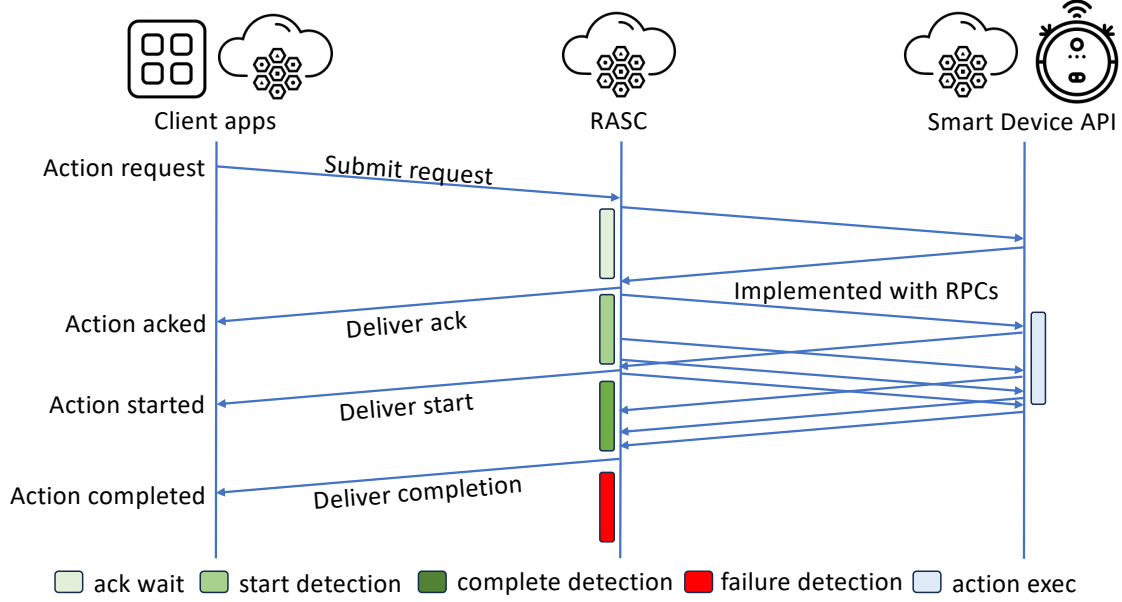


Figure 2.4: **RASC over RPCs**. *Frequent RPCs on right are polls.*

Lifetime of an Action. Figure 2.4 shows how RASC runs backward-compatibly over the RPC layer. For a given action, Rascal cycles through four states (or stages or phases, which we use interchangeably): *ack wait*, *start detection*, *complete detection*, and *failure detection*. When the action is requested, Rascal is in the *ack wait* state. Immediately after the hub receives an ack (that the device or its web service has received the request), Rascal enters the *start detection* phase, wherein it starts tracking the state of the device (described in Section 2.4.1). Alternatively, if the IoT device is unresponsive, then Rascal enters the *failure detection state*. The user specifies a **tolerance threshold** Q_w in seconds (maximum time between a state change and its detection), and Rascal needs to meet it.

If the action is short, the action completion target state might be detected alongside or soon after the start. So during the start detection state, Rascal also checks if either of the start or completion states is already matched, and if so, short-circuits to the complete detection state.

In IoT settings, it is impossible to distinguish a failed device from one that is not executing actions at all. So Rascal detects the failure of *actions*, but does not declare a

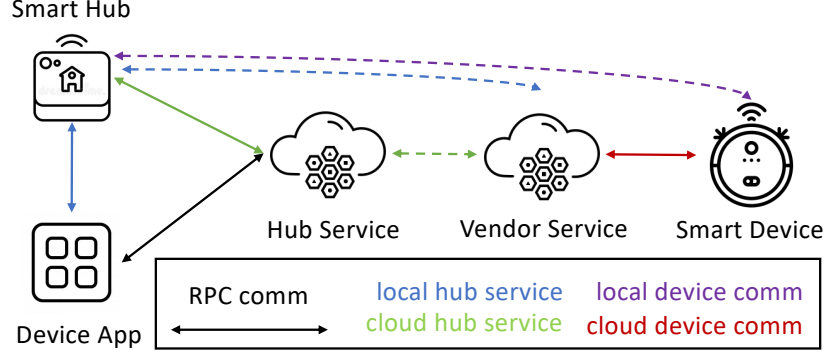


Figure 2.5: **Communication between a smart device and a client app.** *Each arrowed line is an RPC in today’s stacks. This chapter replaces dashed lines with the new RASC abstraction.*

device as being failed. Rascal reports failures to the user (who may take subsequent actions like restarting the device).

IoT Architecture. Figure 2.5 shows how RASC is built atop today’s RPC mechanisms. RASC interfaces user devices and applications (running on the Hub Service, e.g., Alexa, Google Home), with the device or its vendor service (e.g., web microservices run by the device vendor, e.g., Philips Hue, GE Sync, etc.). In Figure 2.5, RASC replaces only those RPCs at dashed arrows, while solid arrow RPCs remain. Every path from the hub/user device to the IoT device contains exactly one RASC call. RASC needs to modify only the cloud services and hubs but not the device or vendor services.

2.4.1 Adaptive Polling Strategy

Both progress updates and failure detection require *polling* that is (i) efficient (few messages) and (ii) responsive (detects changes quickly). For *push* devices (Section 2.2), this is straightforward—the device or vendor notifies the hub, which relays to Rascal. The challenge is *pull*-only devices. Polling too aggressively (e.g., every 500 ms) can overload devices and the network; polling too sparsely increases detection latency. Key questions arise: *What polling frequency is appropriate? Should polling become more frequent as it progresses?*

Problem Statement. We seek to minimize *detection time*: the gap between the action state change on a device and Rascal’s next poll. Inputs are: (i) the **historical**

state-change time distribution $p(t)$ for a (device, action) pair—`dist.pdf(t)`, (ii) a **state change time bound** U —`dist.ppf(0.99)`¹ which may occasionally be exceeded, and (iii) a poll **budget** k representing how many total polls are allowed for a given action (representing allowable bandwidth). We must select k poll timepoints in $(0, U]$ that minimize expected detection time.

Solution. Let the k poll times be $L_i, i = 1, \dots, k$. Then the expected detection time Q can be formulated as:

$$Q = \int_0^{L_1} (L_1 - t)p(t) dt + \int_{L_1}^{L_2} (L_2 - t)p(t) dt + \dots + \int_{L_{k-1}}^{L_k} (L_k - t)p(t) dt \quad (1)$$

$$Q = \sum_{i=1}^k L_i \int_{L_{i-1}}^{L_i} p(t) dt - \int_0^{L_k} t p(t) dt \quad (2)$$

Here, each term i represents the expected detection time of L_i if the change happens before L_i and after L_{i-1} . The highest L_k is always equal to U . The second term here is the average of $p(t)$, and thus a constant we can ignore to optimize Q .

Theorem 2.1 (Adaptive Poll Placement with Fixed Budget k). *Given a time distribution $p(t)$ on $(0, U]$, a poll budget k , and a terminal tolerance $\varepsilon > 0$, polls*

$0 < L_1^ < \dots < L_{k-1}^* < L_k^*$ that minimize expected detection time Q and satisfy*

$|L_k^ - U| \leq \varepsilon$ are given by the following recurrent relation:*

$$L_i^* = \begin{cases} \frac{1}{p(L_{i-1}^*)} \cdot \int_{L_{i-2}^*}^{L_{i-1}^*} p(t) dt + L_{i-1}^* & \text{for } i \in \{2, \dots, k\}, \\ \text{value} \in (0, U] & \text{for } i = 1 \end{cases} \quad (3)$$

¹Percent point function (inverse CDF); `ppf(y)` returns the value with CDF y .

Proof. Per (2), only the i -th and $(i + 1)$ -th terms of Q depend on L_i :

$$A_i = \int_{L_{i-1}}^{L_i} (L_i - t) p(t) dt, \quad A_{i+1} = \int_{L_i}^{L_{i+1}} (L_{i+1} - t) p(t) dt.$$

By Leibniz's rule,

$$\frac{\partial A_i}{\partial L_i} = (L_i - L_i) p(L_i) + \int_{L_{i-1}}^{L_i} \frac{\partial}{\partial L_i} [(L_i - t) p(t)] dt = \int_{L_{i-1}}^{L_i} p(t) dt,$$

and

$$\frac{\partial A_{i+1}}{\partial L_i} = -(L_{i+1} - L_i) p(L_i).$$

The first-order optimality condition $\partial Q / \partial L_i = 0$ gives

$$\int_{L_{i-1}}^{L_i} p(t) dt - (L_{i+1} - L_i) p(L_i) = 0,$$

which rearranges to the recurrence relation (3). Under $p > 0$ and continuity, these conditions determine a unique sequence once L_1 is fixed. We fix $L_1 \in (0, U)$ and generate $L_2, \dots, L_k$ via (3). Then $L_k(L_1)$ is strictly increasing in L_1 . Hence, there exists a unique $L_1^* \in (0, U)$ such that the generated sequence satisfies $L_k^* = U$. $\square$

Next, to find the best sequence $\{L_i^*\}$ we use *binary search*. This leverages our empirical observations that there is a linear relationship between L_1 and L_k . Algorithm 2.1 initially sets the left and right boundaries to 0 and U , respectively. In line 18, L_1 is set to the midpoint of the current left and right values. Using the recurrence equation from (3), lines 20-24 calculate subsequent values based on L_1 . We then check if L_k is close enough to U .² If it is, we return $\{L_i\}$. If not, we adjust the left and right boundaries and recursively search for the correct L_1 .

After we obtain the placement $\{L_i^*\}$, we can examine the second derivative of each L_i ,

²The closeness is measured via the terminal tolerance ε . In our implementation, we set $\varepsilon = 10^{-5}$ (default in numpy `np.isclose()`).

Algorithm 2.1 Adaptive Polling Algorithm

```

1: procedure FINDPOLLS(dist,  $U$ ,  $Q_w$ ,  $slo$ )
2:   return RFindPolls(dist,  $U$ , 0,  $ceil(U/Q_w)$ ,  $Q_w$ ,  $slo$ )
3: end procedure
4: procedure RFindPolls(dist,  $U$ , left, right,  $Q_w$ ,  $slo$ )
5:    $k^* = (\text{left} + \text{right}) / 2$ 
6:    $\mathcal{L} = \text{GetPollingInterval}(\text{dist}, k^*, U, \text{left}, \text{right})$ 
7:   valid = examineQw(dist,  $\mathcal{L}$ ,  $Q_w$ ,  $slo$ )
8:   if left == right & valid then
9:     return  $\mathcal{L}$ 
10:  if valid then ▷ Try to reduce polls
11:    return RFindPolls(dist,  $U$ , left,  $k^* + 1$ ,  $Q_w$ ,  $slo$ )
12:  if N+1 >= right then ▷ Need more polls than right
13:    return RFindPolls(dist,  $U$ ,  $k^* + 1$ ,  $\text{right} \times 2$ ,  $Q_w$ ,  $slo$ )
14:  return RFindPolls(dist,  $U$ ,  $k^* + 1$ , right,  $Q_w$ ,  $slo$ )
15: end procedure
16: procedure GETPOLLINGINTERVAL(dist,  $k$ ,  $U$ , left, right)
17:    $\mathcal{L} = \text{zeros}(k)$ 
18:    $L_1 = (\text{left} + \text{right}) / 2$ 
19:   too_large = False
20:   for i in range(2,  $k + 1$ ) do
21:      $L_i = \text{calculateByRecurrenceRelation}()$ 
22:     if  $L_i > U$  then
23:       too_large = True
24:       break
25:   if isclose( $L_k$ ,  $U$ ) then
26:     return  $\mathcal{L}$ 
27:   if too_large then
28:     return GetPollingInterval(dist,  $k$ ,  $U$ , left,  $L_1$ )
29:   return GetPollingInterval(dist,  $k$ ,  $U$ ,  $L_1$ , right)
30: end procedure

```

where $L_0 = 0$ is a constant:

$$L_i'' = \begin{cases} 2 \cdot p(L_i) - (L_{i+1} - L_i) \cdot p'(L_i) & \text{for } i \in [1, k-1], \\ 2 \cdot p(L_i) + L_i \cdot p'(L_i) & \text{for } i = k \end{cases} \quad (4)$$

If any of them is negative, indicating a maximum instead of a minimum, the placement is invalid and a larger k is required.

Figure 2.6 shows a pictorial example of a PDF for shade up, and the resulting polls generated by Rascal's Algorithm 2.1.

Meeting Detection Tolerance. We need to meet the user-specified *detection tolerance* Q_w (gap between failure and its detection). More critical actions (e.g., involving locks, fire

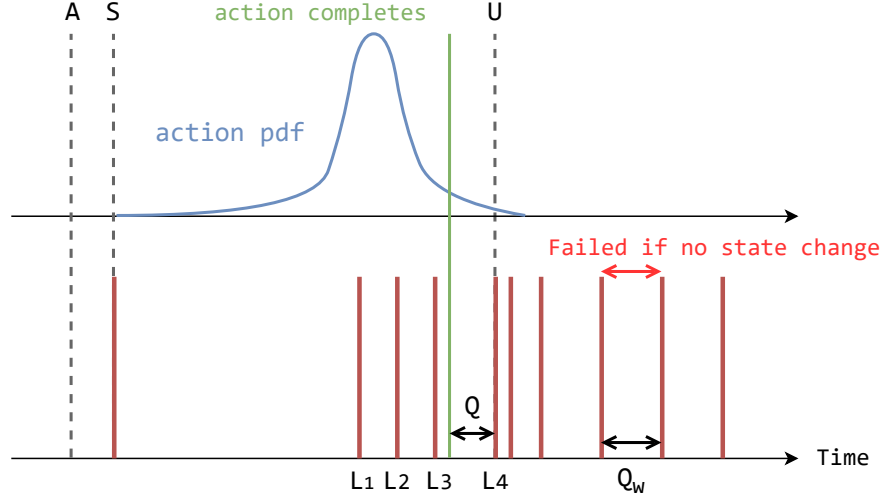


Figure 2.6: **Placement Example**, $k = 4$. Red lines are RASC polls (after action start). Action completes between third and fourth poll.

alarms, exhaust fans) use smaller values (e.g., < 1 s) while less critical actions (e.g., adjusting light brightness, the position of window shades, ovens) can use larger values. We account for this by ensuring that $\forall i : L_{i+1} - L_i \leq Q_w$ and $L_{i+1} - L_i > \text{minimum polling interval allowed by the existing API}$ (naturally, a $Q_w < \text{min polling interval}$ is unsupportable).

Given Q_w , we use binary search to determine the smallest k that meets Q_w —see lines 4-15 in Algorithm 2.1. Line 7 examines the $\mathcal{L} = \{L_i\}$ found by *GetPollingInterval* to make sure any interval between any two polls is smaller than Q_w . If $\mathcal{L}$ is valid and left = right, we return $\mathcal{L}$. If left does not equal right, this means we might be able to find $\mathcal{L}$ with a smaller k^* . The algorithm recursively updates k^* until it finds a k^* that respects Q_w and also cannot be smaller. The algorithm terminates when reducing k^* by 1 would violate Q_w .

Reducing Polling further via Confidence Intervals. We relax the worst-case detection time assumption by introducing a *service-level objective (SLO)*: a confidence level for meeting the detection window Q_w . When SLO=0.9, Rascal’s placement may violate Q_w on at most 10% of events. This allows Rascal to avoid excessive polling in high probability areas of the distribution. Our evaluations (Section 2.7.1) find that SLO-awareness does not change average detection time. Algorithm 2.1 shows how to find polls under SLOs.

Theorem 2.2 (Meeting a Detection Tolerance SLO). *Given a detection window $Q_w \in (0, U]$, a service-level objective (SLO) $slo \in (0, 1]$, and a placement $\mathcal{L} = \{L_1 < \dots < L_k = U\}$, its covered set is*

$$C(\mathcal{L}) = \bigcup_{i=1}^k ((L_i - Q_w)^+, L_i] \cap (0, U], \quad (x)^+ = \max\{x, 0\},$$

and its coverage is

$$\text{Cover}(\mathcal{L}) = \int_{C(\mathcal{L})} p(t) dt.$$

Let $A_k(Q_w)$ denote the best achievable coverage with k polls,

$$A_k(Q_w) = \max_{\mathcal{L}} \text{Cover}(\mathcal{L}),$$

$$k^*(slo, Q_w) = \min\{k \in \mathbb{N} : A_k(Q_w) \geq slo\}.$$

Assume a binary search over k is run with initial bracket **left** = 0, **right** = $\lceil U/Q_w \rceil$, using an oracle **examineQw** that returns **true** exactly when there exists a placement $\mathcal{L}$ with $\text{Cover}(\mathcal{L}) \geq slo$. Then the search returns $k^*(slo, Q_w)$ together with an *slo-feasible placement*.

Proof. Monotonicity. If $k' < k$, any k' -poll placement can be extended to k polls by inserting additional polls; this cannot reduce $C(\mathcal{L})$, hence $A_k(Q_w) \geq A_{k'}(Q_w)$. The oracle predicate is therefore nondecreasing in k .

Bracketing. For $k = \lceil U/Q_w \rceil$, equal spacing $L_i = iU/k$ yields

$\bigcup_i ((L_i - Q_w)^+, L_i] \supseteq (0, U]$, so $\text{Cover}(\mathcal{L}) = 1 \geq slo$. Thus the upper bracket is feasible, while $k = 0$ is not.

Minimality via binary search. Binary search on a non-decreasing predicate with a feasible upper bracket returns the smallest k for which the predicate holds; by definition, this is $k^*(slo, Q_w)$. □

2.4.2 Detection Beyond Upper Bound

Upper bound estimates for action completion may be violated due to human interference (e.g., propped elevator doors), environmental factors (e.g., colder weather slowing a heater), and out-of-order requests (e.g., the elevator moving from 1st to 3rd floor receives a 2nd-floor request). Consequently, if no state change occurs by time U , polling must continue. The key question is: *What is the optimal post- U polling technique?*

Our key observation is that past U , the chance of an imminent state change typically declines, so Rascal polls densely before tapering. We distinguish actions by progress: for those showing *some* partial progress (e.g., vacuum location changed), we estimate the remaining time via the observed rate (time to the current progress) and schedule the next poll at $\min\{\text{estimate}, Q_w\}$. Otherwise, Rascal uses *exponential backoff*, doubling the inter-poll gap up to Q_w , to avoid overloading troubled devices while ensuring detection within Q_w (Figure 2.6). If no change occurs within Q_w after the upper bound U we declare the action failed. In our experiments, overruns beyond U are rare, and late actions typically finish soon after U .

2.5 Building Atop the RASC Abstraction

While Section 2.4 addressed observability, we now turn to programmability: diverse dependencies and dynamic scheduling.

2.5.1 Diverse Action-Causality Expressiveness

Commercial IoT platforms rarely support cross-action dependencies: e.g., Alexa routines allow limited inter-routine chaining,³ but not dependencies between actions. The RASC abstraction enables both cross-action and finer-grained dependencies across critical points *inside* an action’s progress. Figures 2.2a and 2.3d depict *co-action* and *fallback action* examples.

³Alexa can call one routine from another but offers no richer dependencies.

In Rascal each routine is represented internally as a DAG of device actions (Figure 2.2a). When an action emits a progress event (A, S, or C), Rascal inspects its dependent children; any child whose dependencies are satisfied is immediately scheduled to run. Rascal execution is *event-driven*.

2.5.2 Dynamic Action Scheduling

We describe how to schedule routines and actions when some of these actions take a shorter or longer time than expected.

Background. In real-world IoT deployments, action durations vary widely: ovens heat faster when pre-warmed, elevators stall if doors are obstructed, and HVAC cycles fluctuate with outside temperature. Due to these variable times, any static schedule that is made, which devices execute (to respect dependencies) will need to be *dynamically* adjusted at runtime.

The real-time systems community studied this problem through *resource reclamation algorithms* [46], [47], [48]. These algorithms optimize for *early completions*: if a task finishes ahead of its worst-case bound, subsequent tasks can advance to reclaim idle time while maintaining precedence. For example, *Restriction Vectors (RV)* [47] maintains a task timeline and, upon early task completion, pulls dependent tasks forward to begin once all predecessors have completed. This reclaims idle slack while preserving precedence constraints.

Challenges. We need to extend reclaiming algorithms to handle both early completions and actions *exceeding* their schedule. A heater lagging on a cold day or a door closing more slowly than usual delays subsequent tasks, and naive scheduling can break causal dependencies or stall routines.

Rascal’s fine-grained causality model (Section 2.5.1) makes scheduling challenging: we must honor dependencies at arbitrary progress points (e.g., Ack, Start, Complete), increasing potential conflicts and correctness risks.

Correct IoT scheduling hinges on two properties: ***Safety*** (no device receives two action requests at once) and ***Serial Equivalence*** (overlapping routines execute with some serial order). Without them, final states can be unpredictable or inconsistent with any one routine. SafeHome [45] guarantees these only for fixed action durations. However, with unpredictable durations and fine-grained dependencies, maintaining these guarantees requires adaptation.

Overview. Rascal extends reclamation-style scheduling to IoT, building atop the RASC abstraction (Section 2.4). Rascal’s scheduling occurs in two phases: (i) it schedules arriving routines (Section 2.5.2.1), and (ii) it dynamically adjusts the schedule for early state changes and delays (Section 2.5.2.2). We present ***DAG-TL***, to ensure serial equivalence for routines despite variable action lengths, alongside two rescheduling policies, ***STF and RV***, which adapt to both early and late state changes while preserving Safety and Serial Equivalence.

2.5.2.1 DAG-TL: Ensuring Routine Serial Equivalence

We adapt the Timeline Scheduler (TL) from [45]. Upon routine arrival, the original TL places actions in the earliest gaps that preserve intra-routine action order and inter-routine serialization order. TL uses action-by-action backtracking to find gaps that maintain serialization at all times. For Rascal-style DAGs, the original TL has exponential worst-case time complexity.

To handle Rascal-style routines expressed as DAGs, our *DAG-TL* approach adopts an *aggressive backtrack strategy* that makes “big jumps” in the state space. Specifically, upon detecting a serialization conflict, it jumps to the routine’s top-level action and shifts the entire DAG forward in a single step—placing it after the last conflicting time found by inspecting the intersection of the preceding and following routine sets in the serialization order. By pruning unsatisfied paths early, we avoid repeated local retries.

Illustrative example. Figure 2.7 shows three routines: R_A , R_B and R'_B . After sequential

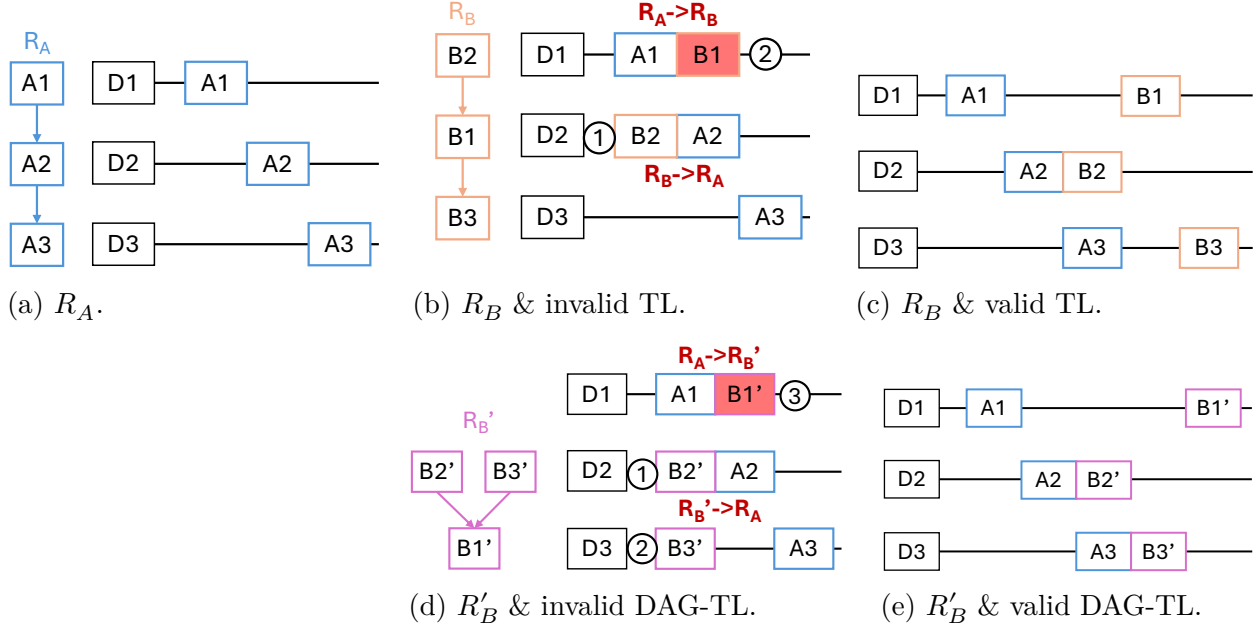


Figure 2.7: **Example Routines and Schedule for TL vs. DAG-TL.** Action symbols feature a letter that stands for the routine they are a part of and a number to denote which device they execute on, e.g., A1 is part of routine R_A and executes on device D1.

R_A is scheduled (Figure 2.7a), sequential R_B arrives. TL fills the earliest gaps: it tentatively places $B2$ (1), then places $B1$ (2), which reveals a serialization violation (Figure 2.7b). TL then moves $B2$ after $A2$ (the next available gap) and completes scheduling R_B without further violations (Figure 2.7c). Now consider non-sequential R'_B arriving after R_A (Figure 2.7d). Here $B1'$ depends on $B2'$ and $B3'$. With a breadth-first traversal, DAG-TL encounters a violation at the third step ($B1'$); it schedules R'_B anew from the root and shifts the entire DAG so that all of R'_B follows R_A on any shared device (Figure 2.7e). Naïve per-conflict backtracking would recheck and reschedule many prior actions and does not scale; DAG-TL's top-level backtrack reduces scheduling steps in practice.

2.5.2.2 Rescheduler

An action may finish later (over-time) or earlier (under-time) than initially scheduled. Rascal must adjust. We describe how actions are rescheduled to accommodate schedule deviations, while minimizing routine completion time and continuing to ensure safety, serial

equivalence, and performance.

Problem Statement. Each routine R is a DAG of actions. An action a has: (i) a device $dev(a)$, (ii) a (possibly data-driven) duration estimate $len(a)$, and (iii) dependency edges to its parents in the routine $Pred_{DAG}(a)$. Multiple routines may run concurrently and touch overlapping device sets.

Our goal is to maintain a *feasible schedule* that (1) is **Safe** (no device runs two actions at once), (2) is **Serially Equivalent** [45] across routines (the final state equals some serial execution of whole routines), and (3) **adapts** when actions finish early/late while preserving (1)–(2).

When to Trigger the Rescheduler. Late actions are handled by Rascal *proactively*, to ensure safety. When a high percentage of the action’s state change length upper bound U (Section 2.4.1) has elapsed (95% in our implementation), Rascal extrapolates to estimate the new state change time (e.g., oven took 20 min to warm up from 200°F to 400°F, then Rascal estimates it will take 5 min more to get to the target 450°F).

Early-state-changing actions are handled *reactively*, as early state changes do not violate safety. If the difference between expected and actual state change exceeds a threshold (1 s in implementation), we invoke the rescheduler.

Constraints on the Rescheduler. Rescheduling must respect two primary constraints: (1) dependency on prior actions in a routine, and (2) the (immutable) serialization order established (Section 2.5.2.1) across active routines. We handle these via preprocessing and two specialized rescheduling algorithms.

Preprocessing. Let $\mathcal{I}$ be the set of potentially impacted actions. We add an action A' to $\mathcal{I}$ when a deviation on action A satisfies any one of three conditions: (i) A' is in the same routine R and is a descendant of A in R ’s action DAG; (ii) A' is scheduled to start after A on the same device D that deviated; (iii) A' belongs to a routine R' that is serialized after R .

To preserve the *established* serial order among routines, Rascal records each routine’s

postset: the routines that, on any shared device, have executed after it up to the rescheduler’s trigger time. It then constructs an immutable serialization order by repeatedly: (1) selecting all routines with empty postsets, (2) placing them at the front of the order (breaking ties by arrival time), and (3) removing them from the remaining postsets. We iterate until all routines are placed.

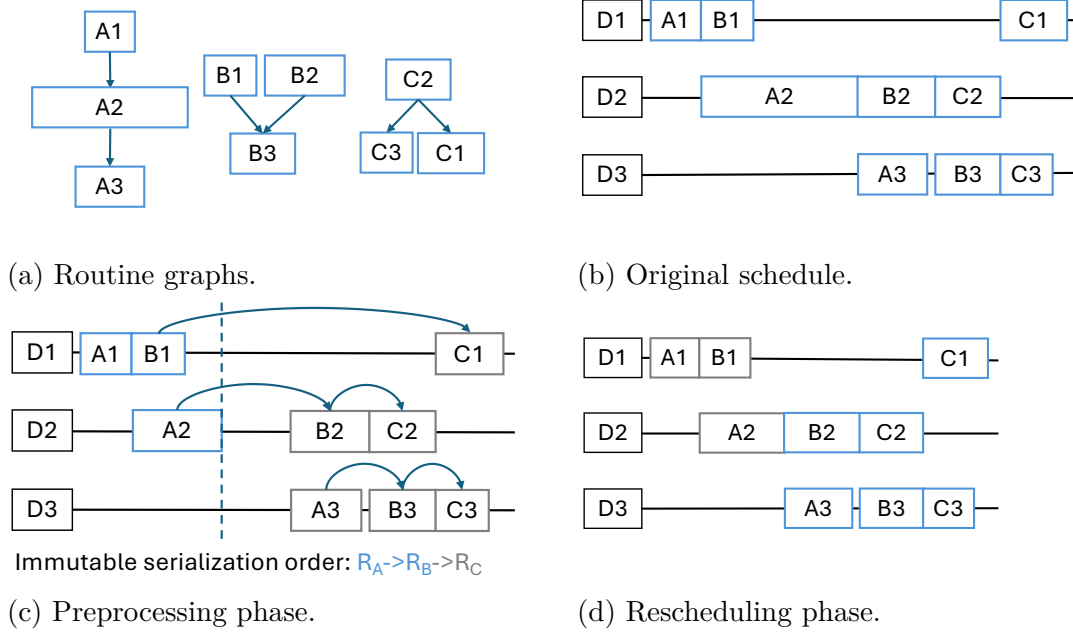


Figure 2.8: **Rescheduling in Rascal.** *Example Routines.*

Rascal deschedules all actions in $\mathcal{J}$, and reschedules them while still preserving the established serial order. For the routines in Figure 2.8a and the original DAG-TL schedule in Figure 2.8b, Figure 2.8c shows the case where $A2$ completes early: affected actions (grey) are descheduled, the immutable order is $R_A \rightarrow R_B$, and routines that have not yet started (e.g., R_C) are left unchanged. Finally, Rascal inserts cross-routine, same-device dependencies to maintain serial equivalence.

Rescheduling. We reschedule descheduled actions using two policies. First, we augment the Shortest Task First (STF) algorithm (known to be optimal [49]) to honor inter- and intra-routine dependencies (Algorithm 2.2). We prioritize actions by estimated length; we repeatedly (i) pick the head (line 11), (ii) place it in the earliest common free slot across

Algorithm 2.2 Shortest Task First (STF) with Serializability

Require: Impacted actions $\mathcal{A}$; duration $len(a)$; device $dev(a)$; routine DAG predecessors $Pred_{DAG}(a)$; immutable routine order $\prec$; device next-free times $next_free[d]$ from current schedule.

Ensure: Updated schedule without violating device safety or serial equivalence.

Phase 0: Add serialization edges (once).

- 1: **for all** pairs of routines $R \prec R'$ **do**
- 2: **for all** devices d shared by R and R' **do**
- 3: Add edge from last action of R on d to first action of R' on d

Phase 1: Initialize readiness and earliest feasible starts.

- 4: **for all** $a \in \mathcal{A}$ **do**
- 5: $Pred(a) \leftarrow Pred_{DAG}(a) \cup Pred_{serial}(a)$
- 6: $indeg[a] \leftarrow |Pred(a)|$; $EST[a] \leftarrow 0$; $FIN[a] \leftarrow \text{unset}$
- 7: $Ready \leftarrow \{a \in \mathcal{A} \mid indeg[a] = 0\}$
- 8: **for all** $a \in Ready$ **do**
- 9: $start_cand[a] \leftarrow next_free[dev(a)]$

Phase 2: List scheduling (single priority queue).

- 10: **while** $Ready \neq \emptyset$ **do**
 - 11: $a^* \leftarrow \arg \min_{a \in Ready} (start_cand[a], len(a))$ $\triangleright$ Earliest start, then shortest task
 - 12: $s \leftarrow start_cand[a^*]$; $f \leftarrow s + len(a^*)$
 - 13: Place a^* on timeline of $dev(a^*)$ at $[s, f)$
 - 14: $FIN[a^*] \leftarrow f$; $next_free[dev(a^*)] \leftarrow f$
 - 15: Remove a^* from $Ready$
 - 16: **for all** successors b of a^* in $Pred(\cdot)$ **do**
 - 17: $indeg[b] \leftarrow indeg[b] - 1$
 - 18: $EST[b] \leftarrow \max(EST[b], FIN[a^*])$
 - 19: **if** $indeg[b] = 0$ **then**
 - 20: $start_cand[b] \leftarrow \max(EST[b], next_free[dev(b)])$
 - 21: Insert b into $Ready$
 - 22: **return** updated schedule
-

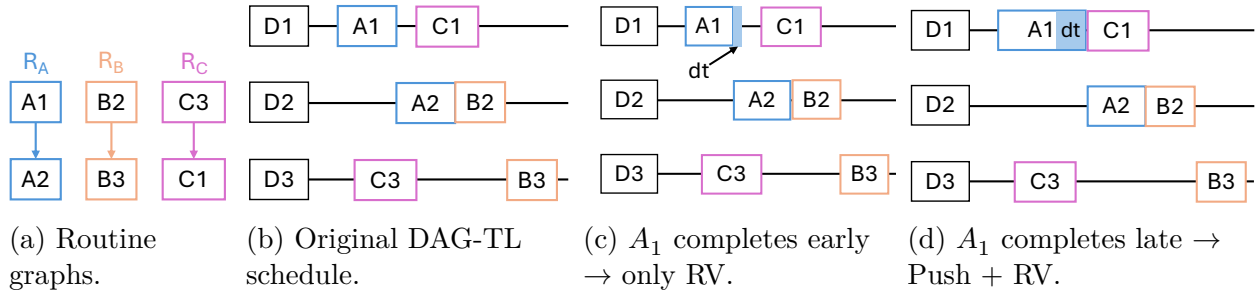


Figure 2.9: **RV rescheduling.** For (d), upon A_1 's late completion detection by an estimated dt , C_1, A_2, B_2, B_3 are first pushed forward by dt before RV is performed.

required devices (line 13), and (iii) update downstream dependencies (line 16), until the queue empties (line 10). In Figure 2.8, STF respects the immutable serialization order, not placing $C2$ before $B2$ despite $C2$ being shorter (Figure 2.8d).

Our second policy adapts *Restriction Vectors (RV)* from real-time systems [47]. The

classical RV pulls a task forward once all its parents finish to reclaim slack. In Rascal, we apply this directly when progress points (e.g., *Start*, *Complete*) occur earlier than scheduled. Unlike classical real-time settings, IoT actions may also finish *later*, which is acceptable; to remain correct, we extend RV as follows: (i) shift *all* subsequent, not-yet-started actions forward by the observed delay to preserve safety; then (ii) apply RV to the shifted schedule to exploit remaining slack. Thus, RV stays correct under delays while still harvesting early-completion gains (see Figure 2.9).

Theorem 2.3 (Action Safety). *For any device D , Rascal: (i) never initiates > 1 action to D concurrently, and (ii) initiates an action only if D is idle. Thus, ack-ed actions on D execute in isolation.*

Proof. *DAG-TL* assigns each action to the earliest idle slot on D that respects dependencies. *RV* may move actions earlier, but only when D is idle and all parents are complete. *STF* inserts each action into the earliest idle slot consistent with serialization. Neither rule permits overlapping initiations. Thus, the non-overlap invariant is preserved under both schedulers and all rescheduling steps. $\square$

Theorem 2.4 (Serial Equivalence). *For any set of concurrent routines $\mathcal{R}$, the schedule $\mathcal{S}$ produced by Rascal is equivalent to executing $\mathcal{R}$ in some serial order $\pi(\mathcal{R})$. This holds for the baseline scheduler and for rescheduling under *RV* and *STF*.*

Proof. *DAG-TL* enforces a partial order $\prec$ on conflicting routines and produces a schedule $\mathcal{S}$ that extends $\prec$. *RV* advances or delays an action's start $s(a)$ only if $\prec$ is preserved. *STF* places each action a in the earliest idle slot $i(a)$ consistent with $\prec$. No step moves an action past the serialized prefix of a conflicting routine; thus, $\mathcal{S}$ is equivalent to some serial order $\pi(\mathcal{R})$ consistent with $\prec$. $\square$

```

1 - id: "1715802493830"
2   alias: Heat when window starts closing and door is closed
3   action:
4     - parallel:
5       - service: cover.close_cover
6         target: { entity_id: cover.living_room_window }
7       - service: cover.close_cover
8         target: { entity_id: cover.balcony }
9     - service: climate.set_temperature
10      data: { temperature: 72 }
11      depend_on: [start, complete]
12      target: { entity_id: climate.main_thermostat }

```

Figure 2.10: **Example RASC routine with `depend_on` specification.** *desc*, *trigger*, *condition* omitted for brevity. The main thermostat is requested to heat to 72°F after the living room window has started closing & the balcony door has closed.

2.6 Implementation

We implemented Rascal as a new component (~8.5K Python LOC) for Home Assistant (HA) [50], a widely used open-source smart-home platform. Deploying Rascal requires no changes to the HA core, integrations, or devices and their APIs. Users simply install Rascal and provide lightweight per-device-class mappings that bind action execution points (i.e., start, completion) to existing device state attributes. This lets Rascal infer action progress and schedule polls without modifying devices. We also built a 500-LOC HA simulator and adapted the `hass-virtual` third-party component [51] for device simulation and Raspberry Pi-based emulation.

RASC API. We extend HA’s action YAML with a single optional field, `depend_on`. This field is an *ordered list* of required progress events—one per parent, in the parents’ order (e.g., `ack`, `start`, `complete`)—enabling fine-grained sequencing. An example routine with mixed dependencies appears in Figure 2.10. If `depend_on` is omitted, we default to `complete`, preserving backward compatibility with HA’s sequential semantics. At runtime, the YAML is compiled into a dependency DAG; DAG-TL monitors this DAG and triggers actions as soon as prerequisites are satisfied.

Action length PDF. We obtain each action’s probability distribution function by storing

historical durations for (ack→start) and (start→complete). Actions are keyed by $\langle device, action, transition \rangle$. Consequently, our dynamic polling strategy requires a brief training phase (several trials) before it can return accurate results.

Asynchronous poll placement computation. Algorithm 2.1’s compute time increases with the upper bound, which can be large for long actions. To avoid polling delays in Rascal, we decouple computation from execution and make it asynchronous: upon an action’s completion, poll placement is computed anew and stored for its next initiation.

Action status change notification. On each status change, Rascal publishes a `RASC_RESPONSE` event on HA’s event bus; applications subscribed to this topic receive the update.

2.7 Experimental Evaluation

We address the following research questions:

1. How efficient is Rascal’s polling technique? (Section 2.7.1)
2. Does action length estimation converge? (Section 2.7.2)
3. If the action length distribution evolves, can Rascal reconverge its learned distribution quickly? (Section 2.7.2)
4. When actions are interrupted, can Rascal distinguish them from a scenario where they actually fail? (Section 2.7.3)
5. Under realistic routines, how do DAG-TL+{RV, STF} compare to baselines [45] in latency & overhead? (Section 2.7.4)

Trace Collection. To drive our trace-driven emulation, we collected traces of (I) device action times that form a diverse set of devices in an office building, and (II) a set of diverse routines, inspired by existing routine datasets including IoTBench [52], [53], described in Section 2.7.4. For (I), we manually collected a significant dataset of action completion

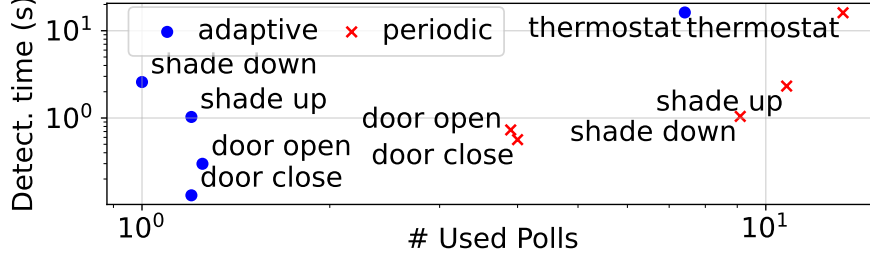


Figure 2.11: **Rascal’s Adaptive Polling vs. Baseline Periodic Polling.** *Rascal is better (lower and to the left) in 4 of 5 cases.*

times. We recorded multiple (> 50) trials over 5 days, with 2 elevators, 2 projector screens, 2 lights, 2 doors, and 4 shades. Elevator data was collected while other users were using it, and thermostat traces are from an existing dataset [54].

2.7.1 Rascal Polling Efficiency

We compare Rascal’s adaptive polling efficiency to a baseline periodic polling strategy, which polls every Q_w . Figure 2.11 shows that compared to periodic polling, adaptive polling uses 44% to 89% fewer polls to detect the action completion. We set the Service Level Objective or *SLO*—the percentage of events Rascal promises to detect within Q_w (Algorithm 2.1)—to 0.9 and Q_w equal to 2, 3, and 30 (seconds) for door, shade, and thermostat actions, respectively. Action **shade down**’s detection time is higher with adaptive polling but still below Q_w . This data point highlights the following tradeoff: periodic polling is costly and guarantees detection within Q_w always, whereas adaptive polling is very cost-efficient but may detect action progress within Q_w only with high probability ($\geq SLO$).

Table 2.2 compares Rascal’s adaptive polling against the baseline called V-optimal (V-opt) [55], the classic dynamic-programming algorithm for histogram construction. V-opt selects bucket boundaries to minimize variance across bins. Both achieve comparable detection times, but Rascal computes poll placements 60-800 $\times$ faster than V-opt. This efficiency gap makes V-opt impractical in real deployments; its computation time can exceed even the average action length (e.g., thermostat), while Rascal completes in

Table 2.2: **Rascal’s Adaptive Polling vs. Baseline V-opt-calculated Polling, Detection vs. Computation Time.** All times are in seconds. **Bold font indicates the lowest value for each action/metric combination.** *Rascal has similar detection times, but V-opt’s computation time is impractical, especially for long actions.*

Action	Avg. Length	Detection Time		Computation Time		Speedup
		Adaptive	V-opt	Adaptive	V-opt	
Door close	3.19	0.19	0.13	6.04e-04	4.38e-02	59.37
Door open	3.06	0.19	0.30	6.43e-04	3.82e-02	72.56
Shade up	29.64	0.31	1.02	3.96e-02	1.97e+01	497.77
Shade down	27.45	1.54	2.58	2.93e-02	1.63e+01	554.99
Therm 68,69	432.17	16.19	–	9.04	2hr+	796.34

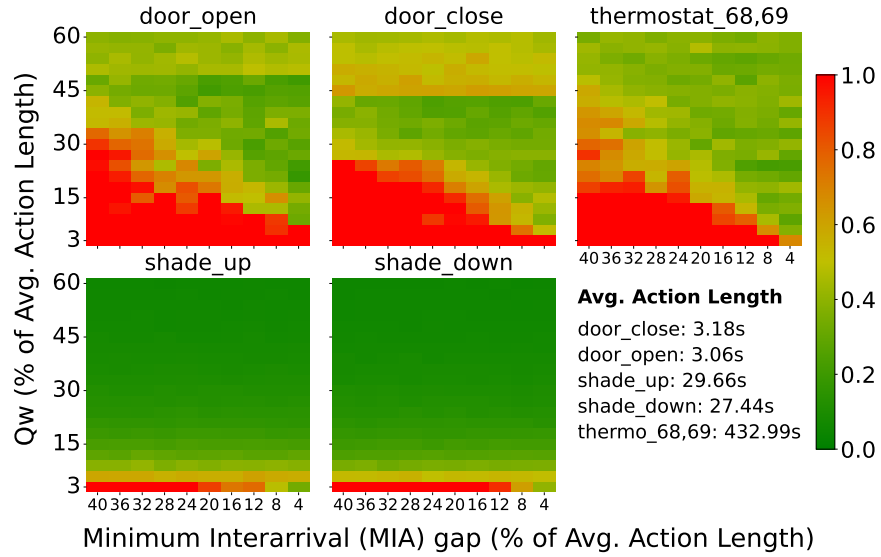
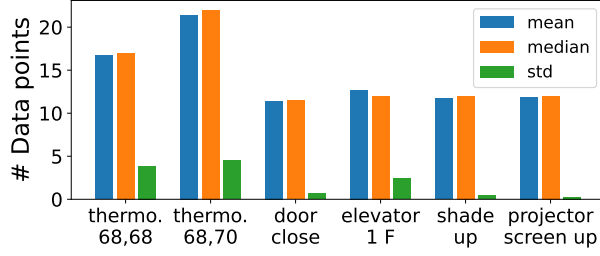


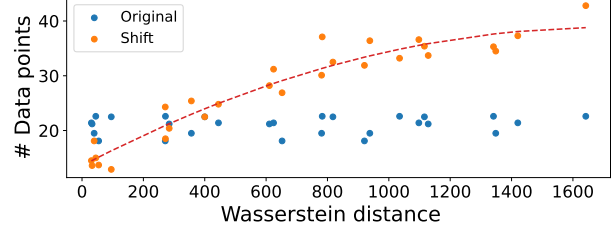
Figure 2.12: **Effect of Extreme Rate Limiting: Rascal’s Detection time under varying Q_w (user-specified tolerance threshold).**

milliseconds.

Figure 2.12 shows the impact of device or API rate limits on Rascal’s polling. The x axis shows the API/device-imposed minimum inter-arrival (MIA) gap between consecutive polls. The y axis shows the user-specified tolerance threshold (Q_w). Both axes are normalized to average action length. The color is $\min(\frac{\text{detection time}}{Q_w}, 1)$. Thus green and yellow indicate detection time threshold being met. We observe that (i) for a given MIA, too stringent Q_w values may not be satisfiable (red in plot); (ii) this Q_w value threshold (for meeting 100% detection: green in plot) drops quickly as MIA becomes less stringent (left to right); and (iii) detection time for uninterruptible mechanical actions (e.g., shade

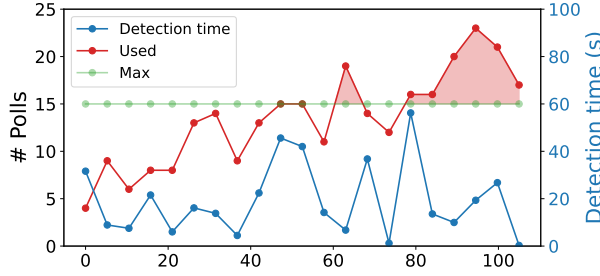


(a) Rascal convergence time.

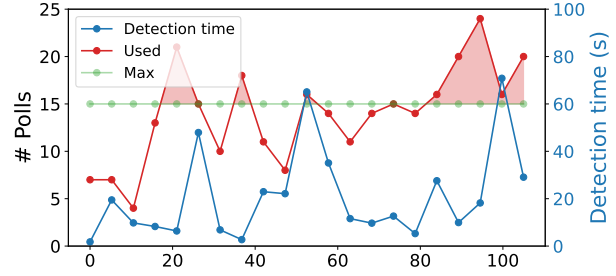


(b) Rascal convergence when action length drifts.

Figure 2.13: **Rascal Convergence.** *Rascal converges quickly, requiring just over 20 (new) data points.*



(a) The interruption happens at the 50% of the action.



(b) The interruption happens at the 80% of the action.

Figure 2.14: **Rascal performance (polls) vs. Interruption Length relative to action length (%).** **Action: thermostat from 68 to 69.** *Rascal generally uses fewer polls than max. Exceptions: when interruptions are long (right side of plots).*

up & down) is more tolerant to rate limiting, because uninterruptible actions have tighter (and thus predictable) action length distributions.

2.7.2 Rascal Training

Convergence Speed. We measure how many action-length samples Rascal needs to reach a *stable* per-(device, action) distribution, i.e., mean and variance both differ by less than 5% upon consecutive samples. Figure 2.13a shows that 10-20 samples suffice across devices: actions lasting tens of seconds (door, elevator, shade, projector screen) converge in ~ 10 samples, while longer actions (e.g., the 400s thermostat) require ~ 20 .

Data Drift. As devices get older, a single action’s probability distributions for its progress points may start moving towards different ranges and probabilities. How quickly can Rascal’s measurements catch up? Figure 2.13b measures (for a thermostat device, at

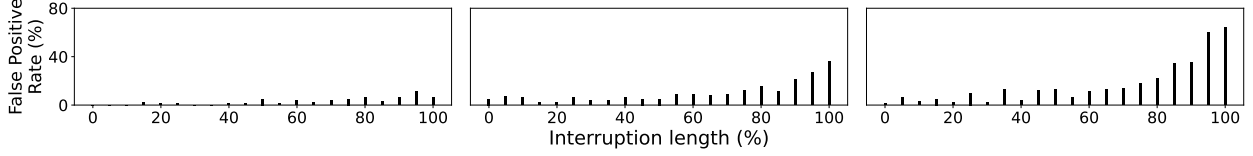


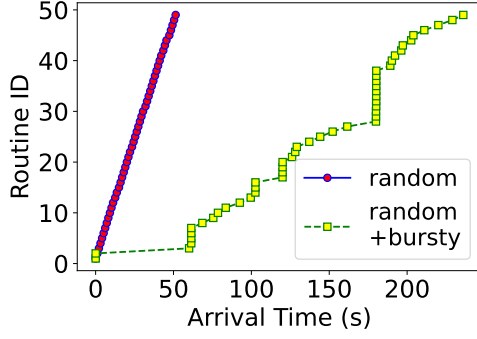
Figure 2.15: **Action Failure False Positive Rate during interruption.** Left to right: **interruption happens (resp.) at the 50%, 80%, and 90% points of action.** *False positives are generally low. Exception: when interruptions are long and towards the action’s end.*

various temperature settings) the Wasserstein (earthmover) distance [56], [57]. This metric measures the distance between the real and learned distributions, for both the original distribution and the shifted distribution. We observe quick convergence, especially when the original distribution is < 500 (already a large value, much higher than the error in our measurements). For instance, the Wasserstein distance between the distributions from the door and the elevator is approximately 13.93. In reality, it is highly unlikely that a door would experience such significant degradation.

2.7.3 Detecting Interrupted Actions

Interruptions to ongoing actions are common due to environmental factors and human interactions. Figure 2.14 shows a real deployment where a thermostat change (68→69) is interrupted at 50% and 80% progress. The red area denotes extra polls beyond U . For early or brief interruptions, Rascal’s overhead is low; with prolonged interruptions (right side of both plots), overhead rises, but detection time (blue line) stays low.

Figure 2.15 shows a simulation where we focus on *False Positive rate*, i.e., the percentage of experiments where Rascal *falsely* detected an action failure. We observe the False Positive rate rises with longer interruptions, especially when they occur near the action’s end, indicating Rascal reliably distinguishes true failures from interruptions in most cases.



(a) The timeline of routine arrivals for the two datasets.



(b) The sum of action lengths per routine as a CDF plot.

Figure 2.16: **Our Routine Benchmark.**

Table 2.3: **Rascal’s resource utilization under different polling strategies. Bold font indicates the lowest value for each dataset/ metric combination. Rascal either incurs the lowest CPU or memory (among baselines), or is comparable to the lowest option.**

Dataset	Polling Strategy	CPU (%)			Memory (MB)		
		avg	q50	q99	avg	q50	q99
random	adaptive	8.79	1.8	19.8	4.12	4.73	8.33
	periodic	9.6	8.6	24.06	3.36	3.33	6.22
	none	2.41	1.1	10.1	17.72	19.19	20.28
random + bursty	adaptive	3.55	1.75	9.68	4.1	4.36	5.9
	periodic	4.6	2.9	19.26	4.54	4.61	6.58
	none	2.72	1.3	22.39	17.87	18.21	19.07

2.7.4 Routine Scheduling

We evaluate Rascal’s end-to-end performance: overheads and comparing DAG-TL scheduling (Section 2.5.2) to baselines.

Dataset of Routines. We generated a suite of routines spanning diverse devices and lengths—statistics of our routine dataset are shown in Figure 2.16. We have two arrival workloads: (1) *random*, attempting to capture “background” routine arrival throughout the 24 hours of a day, and (2) *random+bursty*, a mix of 50% random and 50% bursty arrivals, intended to capture “peak” times of activity, such as morning, lunch, evening.

Overheads. Table 2.3 reports Rascal’s process overhead under different polling strategies. Adaptive polling is consistently more efficient than periodic—up to 63% lower CPU and

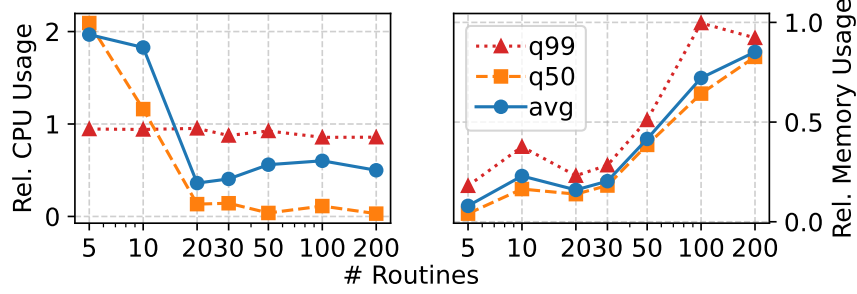


Figure 2.17: **Rascal’s relative resource utilization across varying routine loads.** *Rascal’s CPU overhead is consistently low, while memory usage scales sub-linearly with the # concurrent routines.*

43% lower memory. Disabling polling (*none*) yields only marginal or inconsistent gains (CPU q99 up to 68%, otherwise up to 33%), and memory can be worse. In fact, memory is higher with no polling than with Rascal, indicating our automation-tracking memory management outperforms Home Assistant’s baseline, which overuses external libraries.

Figure 2.17 illustrates Rascal’s *relative* resource utilization, calculated as the ratio of utilization with *adaptive* polling to utilization with polling disabled (*none*). We vary concurrent routine load from smart home scale (up to 10) to smart buildings and campuses (up to 200). Both Rascal’s 99th percentile CPU & all memory metrics stay below that of disabled polling. While mean (median) CPU usage peaks at $\sim 2\times$ at low loads (up to 20 routines) due to fixed scheduling upfront Rascal overhead, it then drops, stabilizing at $\sim 0.6\times$ ($\sim 0.1\times$). Relative memory rises with load but plateaus with 200 routines at $\sim 0.8\times$. This is the resource vs. benefit tradeoff entailed by Rascal.

Figure 2.18 shows Rascal’s rescheduling overheads with STF (Shortest Task First). For both datasets, rescheduling time drops as action-length estimates increase, indicating that overestimation reduces overhead—even with more early completions. The random+bursty dataset achieves faster rescheduling time as the schedule is tighter at times and traversals are faster.

Rascal Rescheduler Vs. Baselines. We compare two Rascal reschedulers (STF and RV, Section 2.5.2.2: labeled in our plots as DAG-TL+STF and DAG-TL+RV respectively) against three baselines. Our three baselines are: (i) FCFS (First Come First Served): strict

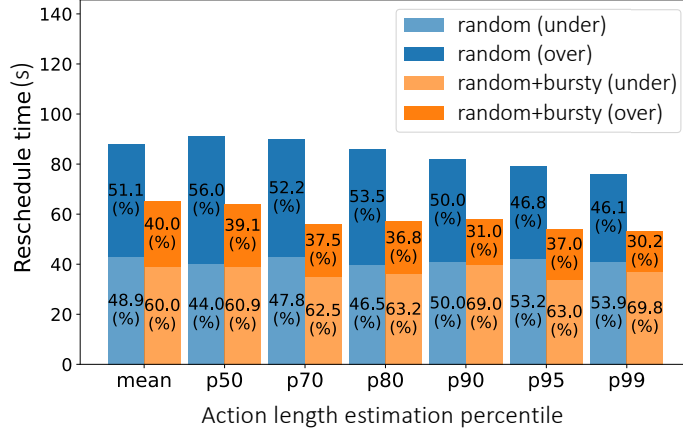


Figure 2.18: **Rescheduling Overhead.** Bars show total rescheduling time over the dataset, split into under-/over-time handling. Rascal overhead stays low.

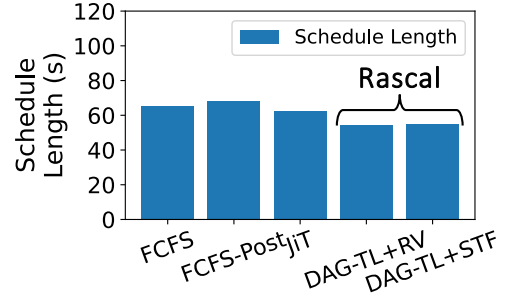


Figure 2.19: **Schedule length comparison.** Rascal (right two) creates at least 11% faster schedules vs. Baselines (left three).

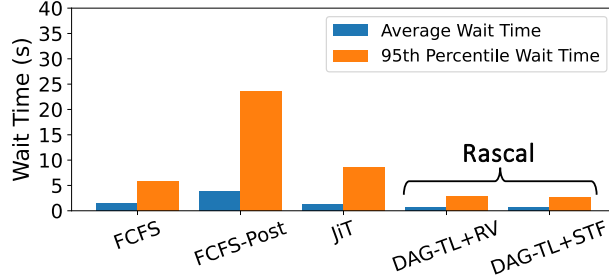


Figure 2.20: **Wait time comparison.** Rascal (right two groups) decreases wait times by at least 33% and 50% for mean and q95 values, respectively, vs. Baselines (three left groups).

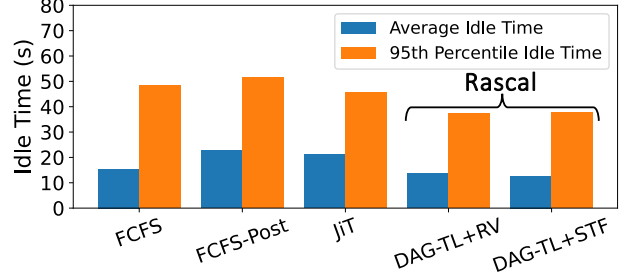


Figure 2.21: **Idle time comparison.** Rascal achieves at least 27% and 18% lower mean and q95 values, resp.

no-overlap scheduling by arrival; (ii) FCFS-Post: a faster FCFS variant allowing a routine to start once predecessor routines finish on all shared devices; (iii) Just in Time (JiT) algorithm [45], a greedy, opportunistic approach.

Figures 2.19 and 2.20 show that (i) Rascal’s two variants (DAG-TL+RV and DAG-TL+STF) perform similarly, (ii) Rascal’s slowest variant (DAG-TL+STF) finishes routines 11% faster than the fastest baseline (JiT), and (iii) Rascal’s slowest variant’s wait times are also lower—mean by 33% and q95 by 50%—than the strongest baselines (JiT and FCFS, respectively). Figure 2.21 shows that Rascal reduces idle time (time a device is not utilized) by 27% at mean and 18% at 95th percentile compared to the fastest baseline (FCFS and JiT, respectively). In Figure 2.22, Rascal’s slowest variant increases the routine

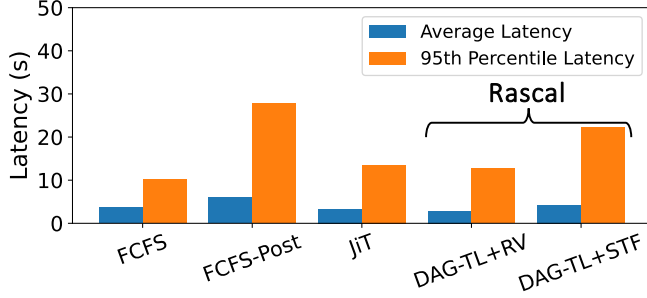


Figure 2.22: **Latency comparison.** *Rascal achieves at least 25% and 55% lower mean and q95 values, resp.*

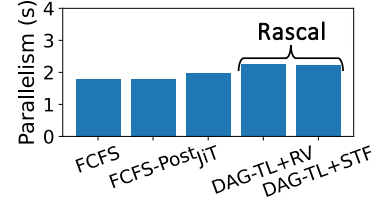


Figure 2.23: **Parallelism comparison.** *Rascal achieves > 10% parallelism.*

end-to-end latency average time by only 25% and 95th percentile by 55%, compared to the fastest baseline (JiT and FCFS, respectively). Finally, Figure 2.23 shows that Rascal’s least efficient variant achieves 10% higher average parallelism rates over the most efficient baseline (JiT). Overall, Rascal’s rescheduling improves metrics by 10–55% over baselines.

2.8 Discussion

Action-length predictability and richer observability. Action lengths of some devices (e.g., shades) are more predictable than others, e.g., a thermostat (HVAC) may have seasonal variations, oven cook time depends on food, etc. Section 2.7 evaluated *representative* devices that span this spectrum. More expressive observability (e.g., progress bars) may be feasible atop RASC, but it must be *simple* and *accurate*.

Causality expressiveness. Current automation interfaces (e.g., Alexa [29], Google Home [30]) support limited causality expressiveness. Their IFTTT-style [22] rules struggle with conditional, multi-stage dependencies. Future work could integrate expressive, flow-based (and visual) systems like Node-RED [23] or Flogo [58].

Privacy. RASC does not alter the privacy posture of the deployment it runs in. All of its polling, lifecycle-state tracking, and scheduling operate within the same administrative domain as the underlying platform (e.g., a home running Home Assistant [39]): RASC introduces no new data flows that cross the domain boundary, sends no telemetry to third parties, and observes only device state that the platform already collects. A deployment

that keeps its control logic on-premise therefore retains exactly the same privacy boundary with RASC as without it. Domains with multiple users at differing authorization levels—e.g., a smart factory where contractors are only partially authorized while employees have full access—will already have privacy and access-control mechanisms of their own. These are orthogonal to RASC, but integrating with them (so that observability is granted only over the actions a given principal may see) is a prerequisite for deploying RASC in such settings.

Scaling to many users. Smart spaces served by RASC range from homes to commercial buildings, and larger spaces have correspondingly more users. For RASC, additional users manifest primarily as *scale*—more devices under management and more concurrent actions and routines—which is exactly the dimension Section 2.7 evaluates. Multiple users also raise concerns that are orthogonal to RASC’s abstraction, such as the privacy and authorization questions discussed above; addressing those is left to future work.

Maintaining conditions over long periods. Some automation goals are *standing* rather than one-shot—e.g., keeping the floor clean when a pet drops food repeatedly through the day. Such goals are typically programmed as one or more sensor triggers paired with one or more routines, each invoked whenever a trigger fires, and are thus orthogonal to RASC’s contributions: RASC makes every invoked routine observable, whatever triggers it.

Exposing tradeoffs and costs to users. The tolerance threshold Q_w (Section 2.4) lets users trade detection latency against polling cost, but the value a user picks then applies uniformly: RASC offers no way to *adapt* it to context. Different contexts warrant very different settings. Our evaluation used $Q_w = 2$ s for door actions, 3 s for shades, and 30 s for thermostat actions (Section 2.7.1), and a household might reasonably want its front door watched at $Q_w = 0.1$ s while the house is empty but only at 10 s while someone is home. Those two settings differ sharply in what they cost: at 0.1 s RASC must issue many more polls and, under the minimum inter-arrival gaps that device APIs impose, may not be able to meet the threshold at all (Figure 2.12), whereas 10 s is satisfied with far fewer polls

and correspondingly less network and energy overhead. Supporting per-context thresholds—and showing what each setting costs before it is chosen—would let occupants tune RASC to their priorities as those priorities shift through the day.

2.9 Related Work

Push-based Frameworks. RASC shares semantic roots with ROS (Robot Operating System) actionlib [59], gRPC streams [60], and MQTT [61], which provide goal-feedback-result lifecycles or push-based notifications. However, these frameworks typically require device-side protocol changes. RASC is minimal.

Scheduling and Action Dependencies. A range of systems provide abstractions for coordinating IoT and cyber-physical devices. Gaia [62], Bundle [63], dSpace [64], and DepSys [65] offer middleware or dependency-based programming, but at coarse granularity without action-level progress or dynamic scheduling. TransActuations [66], Rivulet [67], and IoTRepair [68] strengthen safety through transactional or rollback semantics, yet operate only at the level of instantaneous or fixed-length actions or routines.

SafeHome [45] and Hades [69] handle routine conflicts but not fine-grained dependencies. In contrast, Rascal captures dependencies within as well as across routines, and our schedulers ensure safety and serial equivalence even as actions progress unpredictably.

Dynamic Rescheduling. Classical resource-reclaiming in real-time systems [46], [47], [48] captures slack from early tasks but assumes fixed DAGs and ignores late completions. Our adaptation of RV and STF extends these ideas to IoT action graphs with cross-routine serialization constraints, handling both early and delayed completions without violating safety.

Observability and Failure Detection. IoT systems often rely on coarse timeout-based detection or assume instantaneous actions [70], which is unsuitable for long-running tasks. SafeHome [45] and Hades [69] provide some exception handling but not progress-aware detection. Our adaptive polling provides fine-grained observability of start/complete

events, enabling efficient and accurate failure detection without overloading the system. Complementary HCI work surfaces this uncertainty to people rather than to the system: predictive displays for teleoperation under delay externalize *when* a command takes effect and how inputs map to motion [71], mirroring at the human interface the action-progress visibility Rascal provides at the systems level. While pub/sub systems [72], [73], [74], [75] are often proposed as alternatives, they are best suited for settings with many subscribers across diverse topics. Rascal runs on a single central hub that subscribes to all topics, so pub/sub offers little benefit, and polling remains the core challenge it addresses. The histogram bucket problem [76], [77], [78], [79], [80] is analogous: both tasks require selecting partition ranges (bins vs. polling intervals) under uncertainty, differing only in the optimization criterion—e.g., minimizing error in density estimation versus balancing observability cost and timeliness.

Commercial Systems. Commercial platforms such as Alexa [29], Google Home [30], SmartThings [81], and Home Assistant [39] support limited forms of automation and do not guarantee conflict freedom or robust failure handling. RASC generalizes them for expressive DAG-based routines, with progress awareness and principled scheduling guarantees.

2.10 Conclusion

RASC provides an expressive RPC alternative for IoT device collections such as smart homes, smart buildings, etc. Our implementation, Rascal, supports *observability* by detecting action completion and failures quickly, and *programmability* via fine-grained causality dependencies, and assures safety and serial-equivalence. Rascal requires no modifications to existing devices. Trace-driven evaluation showed that Rascal detects completion within 2–13 RPCs and 2–16s over 90% of the time, even for actions lasting tens of minutes. Our routine scheduling outperforms state-of-the-art baselines by 10–55%.

Chapter 3 CoMesh: A Decentralized Control Plane for Commercial Edge IoT Meshes

This chapter targets large commercial edge deployments—smart buildings, farms, and factories—in the smart space taxonomy of Table 1.1, addressing the *smart-space automation plane* of smart-space control introduced in Section 1.1: *where* should control logic run? We present **CoMesh** [15], the first fully decentralized, hub-less control plane for Sense-Trigger-Actuate (STA) operations in large commercial edge IoT mesh networks. Rather than routing coordination through a central hub or cloud, CoMesh keeps all decision-making local to the mesh by organizing smart devices into overlapping subgroups called *coterie*s that collectively monitor devices and manage routines via zero-message exchange protocols, quorum-based agreement, epoch-based migration, and locality-sensitive hashing (LSH). The result is a self-healing architecture that adapts autonomously to device failures, churn, and mobility without any human intervention or centralized oversight, realizing a vision of edge-first smart environments that remain fully operational without external infrastructure.

This chapter is organized as follows. Section 3.1 introduces the problem and surveys the commercial edge landscape; Section 3.2 presents CoMesh’s design overview. Section 3.3 defines the system model. Section 3.4 describes CoMesh’s coterie mechanism, including zero-message member selection, quorum-based agreement, epoch-based migration, and LSH-based locality. Section 3.5 covers device and routine management, including conflict-free actuation via distributed locks. Section 3.6 provides formal proofs of CoMesh’s safety, liveness, and progress properties. Section 3.7 reports our Raspberry Pi deployment results and Section 3.7.2 presents large-scale simulation results using real building maps. Section 3.8 discusses related work and Section 3.9 concludes.

3.1 Introduction

Commercial edge deployments with Internet of Things (IoT) devices are rapidly growing in size and density. There are around 21 B connected IoT devices today, forecast to reach roughly 40 B by 2030 [1]. Commercial settings are the next frontier beyond smart homes (Figure 1.1). For example, smart buildings already account for around 2 B devices, and are forecast to reach 4.12 B devices by 2030, with the building IoT market exceeding \$100 B [5]. Other emerging areas for IoT deployment in commercial edges include smart farms with sensors and robots [3], Industry 4.0 [4], battlefields [82], and indoor spaces like warehouses [83]. Each of these market sectors is growing fairly quickly [84].

Because commercial settings have large physical footprints (unlike smart homes, which are small), it is common for them to contain *mesh networks* among the IoT devices. Mesh networking technology has matured using Wifi, Zigbee, Bluetooth, LoRa, etc. [85]. Industry commonly uses IEEE 802.15.4, WirelessHart [86], or IEC [87] to build such IoT meshes. A mesh network (a.k.a. an ad-hoc network) is robust, resilient, scalable, and enables control without requiring a central operating unit (or hub), which may be prohibitive to set up, be overloaded, or prolong latencies. These mesh networks usually contain a mixture of devices that are “smart” devices capable of (small amounts of) compute and storage, along with “simple” devices that can only receive commands. There is rich literature at the networking layer, e.g., on how to build, configure, and perform routing in mesh networks [88], [89].

Sense-Trigger-Actuation of Routines. Sitting *above* this mesh network is the *control plane*—this typically needs to execute three kinds of actions: *Sense-Trigger-Actuate* (or STA). The control plane needs to first *sense*: continuously collect measurements from multiple devices. Applications specify predicates, which are arbitrary Boolean clauses involving multiple device readings. Whenever a predicate is satisfied, the control plane needs to *trigger* a programmed series of actions, sending commands to several devices. Third, because some of these actions may be long-running (e.g., mechanical movements, repetitive actions, or time-based actions such as water sprinklers), the control plane needs

to initiate and monitor *actuation* of these commands (at multiple devices) quickly and efficiently. The key challenge is to actuate in an order that *avoids conflicting actions on devices* [66], [90].

This STA abstraction is commonly programmed via *routines*, available in Samsung SmartThings [81], Amazon Alexa [29], Google Home [30], etc. A routine is a sequence of *commands* that actuates multiple devices, wherein each command executes an action on one device. Multiple routines may be active at any given time, while others remain dormant (waiting to be triggered). A routine may be triggered either based on sensor values (Boolean predicate), or at particular times, or a combination of these, or manually by a human.

CoMesh reuses these familiar routine front-ends rather than introducing a new programming interface. Figure 3.1 shows CoMesh’s programming workflow: a user authors a routine through an existing graphical user interface (GUI) such as the Samsung SmartThings app, which is exported as SmartThings’ internal JSON; CoMesh converts this to its own JSON specification, which is then promulgated to the smart devices that monitor and execute the routine. CoMesh likewise integrates with Amazon Alexa and Google Home. Thus the abstraction a user sees is unchanged; CoMesh only changes *how* the routine is managed underneath—in a decentralized, hub-less fashion.

This chapter focuses on management of such routine-based workloads running over large commercial edge IoT meshes. Figure 1.1 (from Section 1.1) shows three emerging edge mesh scenarios—a smart building (b), a smart farm (c), and a smart factory (d). Some examples of routines follow. A smart office will have routines running day and night, e.g., to: (i) raise and lower shades and thermostat, in response to sunlight and temperature; (ii) change interior lighting as a function of occupancy; (iii) enable security cameras and lock gates at 9 pm. A smart farm has routines that: (iv) water dry areas of the farm when soil sensors detect dryness below a threshold; (v) trigger robots to inspect crops. Routines might be triggered by sensors, time (fixed or periodic), or humans. For a

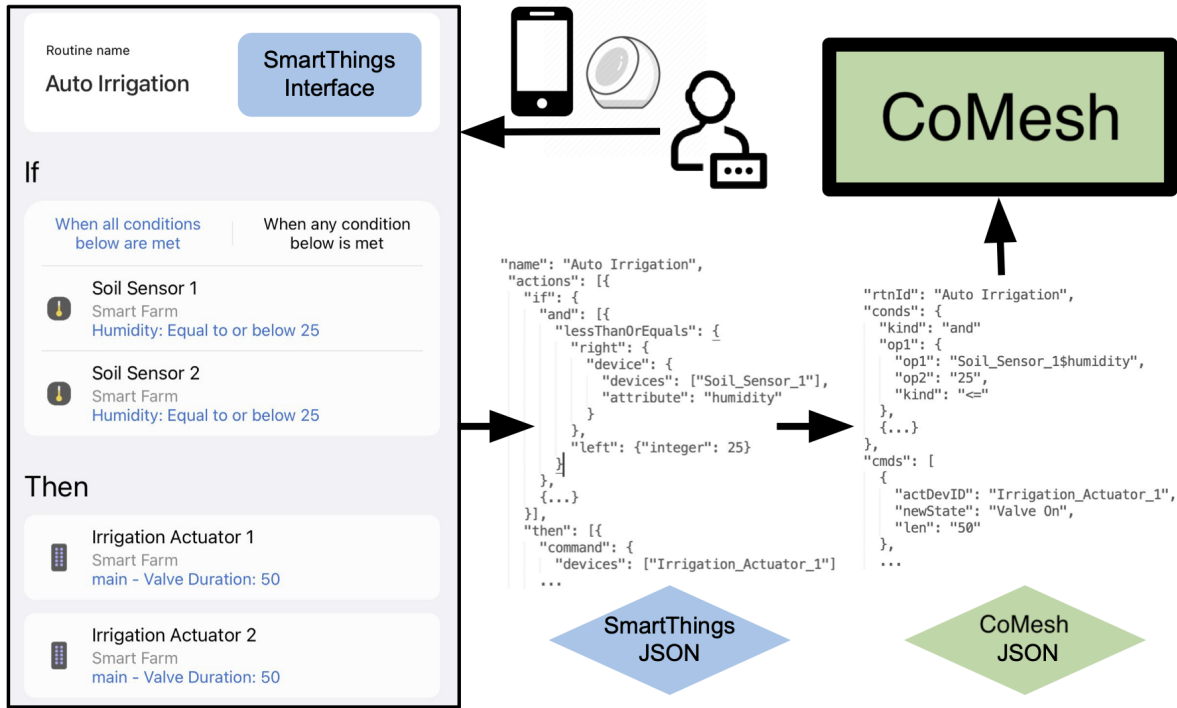


Figure 3.1: **CoMesh Programming Workflow.** Routines are authored in an existing GUI such as the Samsung SmartThings app, and SmartThings’ internal JSON is converted to CoMesh’s JSON specification before being promulgated to devices.

routine that is sensor- and/or time-triggered, its trigger clauses need to be monitored continually so that after the trigger becomes true, the routine starts execution *quickly*.

Another key challenge in scheduling these routines is to avoid *conflicts among routines*, i.e., *two routines that actuate intersecting sets of devices should not be allowed to execute concurrently, but instead be executed serially*. Handling conflicts is an important and difficult problem, but existing solutions [90] are centralized and hub-based.

Towards Cloudless and Hubless STA Operations. Commercial edge settings have a strong requirement of uninterrupted operation. This has led to recent efforts that build “Cloudless IoT” solutions across both industry [91] and open source [8], [9]. Benefits of cloudlessness include: lower latency, stability (cloud outages do not affect local operations), persistence (no effect of changing cloud services and APIs), lowered cost for providers (no cost of running on cloud services⁴), and keeping operations+data private and local.

⁴Provider cost for a customer with 500 devices pinged every second, plus routine actuation, can be as high as \$2500 per month [11].

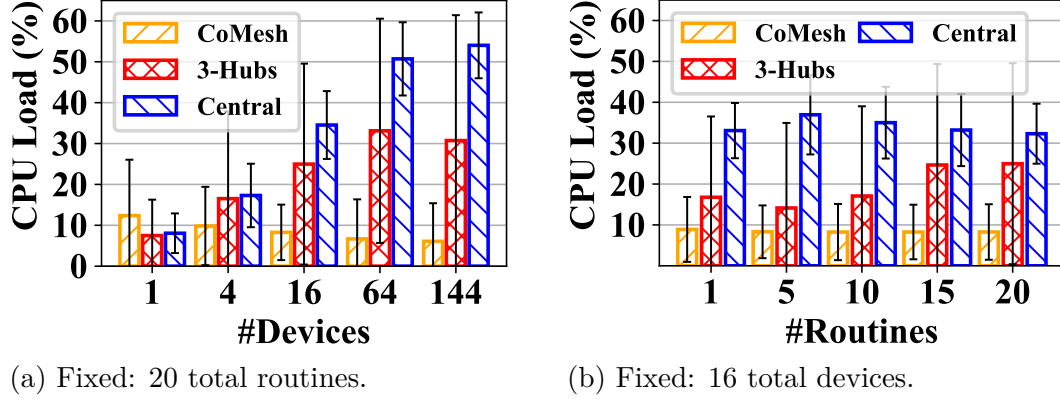


Figure 3.2: **Raspberry Pis CPU utilization.** *Our decentralized CoMesh reduces CPU utilization significantly compared to the centralized Hub and 3-Hubs.*

A cloudless commercialized edge has two possible design paths: (1) a local hub-based solution, or (2) a *hubless* (fully-decentralized) solution. Preliminary work exists on the former [8], and the latter [67]. Rivulet [67] is the first *decentralized* (hubless) IoT solution we know of, and it is scoped on data streams and individual commands, *not* routines. No existing system handles inter-routine conflicts, *and* with decentralization. This chapter takes the next big step in making cloudless commercial edges hubless and fully decentralized. The cloudless+hubless approach addresses concerns of the hub being overwhelmed (Figure 3.2), and being susceptible to unreliability [92] and cloud outages [10]. Figure 3.2(a, b), done with Raspberry Pis, shows that with more devices (i) CPU load on the central hub rises rapidly and crosses 50% at just 64 devices, (ii) even with 3 central hubs sharing the load, the CPU load is high (30%) and has high variance. The per-device CPU load is reduced by over $6\times$ (beyond 64 devices, or 15 routines) vs the 3-hubs by our decentralized solution (CoMesh), at 5.2%-12.4%, and it scales (remains flat). Regardless of load concerns, hubless solutions are paramount due to needs of user privacy (of data & operations), cloud outage resilience, and lowering cloud costs.

Table 3.1 details the tradeoffs between CoMesh and an alternate solution that might use multiple (N) hubs to coordinate device and routine actions. Overall, N -hubs can be simpler as it uses static sharding, but this requires manual reconfiguration, e.g., when devices are added/removed, hubs are added/removed. CoMesh dynamically reconfigures

Table 3.1: N-hubs vs. CoMesh design trade-offs.

Property	N-Hubs (Sharded/Hierarchical)	CoMesh (this chapter)
Complexity	Simpler at small scale	Complex, but with correctness proofs
Device Impact	Processing only at hubs	Devices process, but load is balanced
Deployment, Setup & Scalability	Static sharding	Spatial rebalancing via coteries
	Manual reconfig via static sharding; dynamic reconfig needs CoMesh-like logic	Dynamically reconfigurable with churn
	Hubs can overload	Temporal rebalancing smooths long-term load
	Scalability limited by #hubs & inter-hub coordination	Scales well
Fault Tolerance	Limited resilience; hub faults disrupt many routines	Tolerates up to f device faults via quorums/migration
Resource Overhead	Preferable for small-scale setups	Preferable for larger on-field setups with many devices

and scales arbitrarily, but comes with some level of complexity; we balance this complexity with proofs of correctness. At its core, however, autonomous coordination among N hubs *will* require coordination similar to CoMesh’s mechanisms. In other words, one can reuse CoMesh’s core techniques to also implement a solution with N hubs, and be assured of correctness.

A cloudless+hubless STA solution requires that all actions: (i) operate *over the local edge mesh*, (ii) remain *local*, without requiring a connection to the cloud, and (iii) maintain *fault tolerance*⁵ in spite of local device failures.

Contributions. We present CoMesh, the first system to realize a *fully decentralized* (i.e., hub-less and local) control plane for monitoring and management of routines over large edge meshes. The contributions of this chapter are:

- Design and implementation of CoMesh, using a novel combination of techniques: (1) distributed sense, trigger, and actuate operations over both space and time; (2) fast coordination via zero-message exchange and quorum-based protocols; (3) distributed coordination to ensure conflict-free actuation of routines; and (4) locality-based hashing (LSH) to keep communication local. While some of these techniques are

⁵While this thesis only deals with crash failures, our decentralized approaches spread the attack surface and are more secure than a centralized cloudless solution. Security is beyond the scope of this thesis.

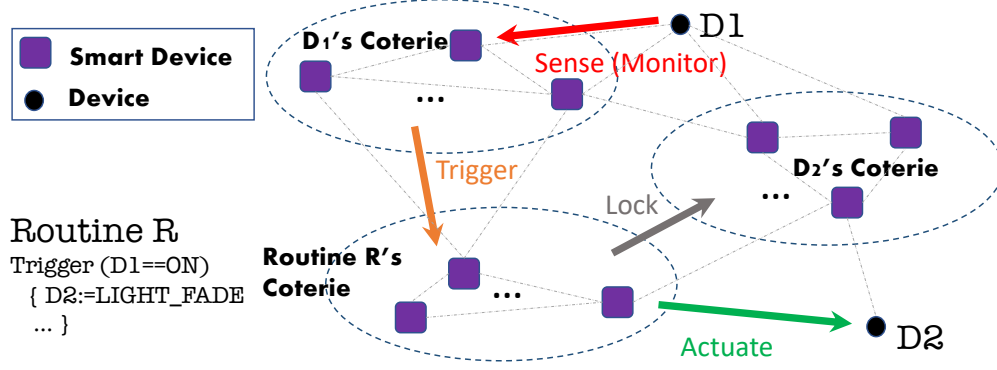


Figure 3.3: **Control Flow of CoMesh’s Sense-Trigger-Actuate Actions.** *Ad-hoc network links are shown in the background. (For simplicity, all coterie are shown as hyper-local.)*

known, CoMesh is unique in the way it combines them to solve STA challenges.

- Our Raspberry Pi-4 deployment experiments and simulations, using real building maps and real routine workloads, show CoMesh is scalable, efficient, fault-tolerant, and fast.

Our source code and datasets are available at

<https://dprg.cs.uiuc.edu/traces/go.php?id=42> and

<https://dprg.cs.uiuc.edu/traces/go.php?id=43>.

3.2 Challenges and CoMesh’s Overview

Challenges of Decentralization. CoMesh’s goal is to design a hub-less, fully decentralized, local, STA management system that relies on neither a single central hub nor a cloud. CoMesh contains *local* distributed protocols running among smart devices which have compute and memory resources (other simple devices without such resources may also co-exist)—CoMesh requires only a fraction of devices to be smart. Smart devices may include IoT devices, Raspberry Pis, home hubs, handheld devices (phones, tablets), etc.

CoMesh has to tackle five major challenges: (1) *fully decentralized* operation: sensing (monitoring), triggering, and actuation of routines, without any reliance on centralization; (2) *conflict-free* actuation of routines: no two routines that actuate intersecting sets of devices should be simultaneously active [66], [90]; (3) *load balance*: of STA overhead both

across devices and across time; (4) *robustness*: to tolerate failures of devices; and (5) *physical awareness*: network load that is aware of the physical layout of devices.

CoMesh’s Overview. Now we summarize key internal components of CoMesh. Internally CoMesh balances the STA load across both *space and time*. Spatial balancing divides the STA load (of an erstwhile hub) across multiple smart devices. Since some devices may be more popular (appear in more routines), and some routines run more frequently, temporal balancing spreads the load from these “popular” devices and routines across smart devices. CoMesh implements such balancing by using *coterie*s—subgroups of smart devices responsible for devices and routines. For *Spatial balancing* there is: (i) a separate coterie for each device, responsible for continually sensing that device, and (ii) a coterie per routine to handle its triggering, actuation, and conflict-free execution. Figure 3.3 depicts that a routine’s coterie coordinates with the coterie of all its trigger-clause devices as well as coterie of its actuated devices, in order to ensure conflict-free operation of routines. To reduce coterie count further, CoMesh multiplexes multiple devices/routines into the same coterie. This means that a given device may *belong* as a member of multiple coterie, but there is still exactly one coterie to monitor a given device, and exactly one coterie to manage a given routine.

Coterie selection comes in two flavors—hashing-based, and locality-sensitive-hashing based which keeps coterie communications local. Within each coterie we carefully design fast and cheap IoT coordination via quorum-based agreement, leader election, and tolerance to coterie leader and member failures. Specifically, to tolerate f failures in the system, it is sufficient for each coterie to contain $k = (2f + 1)$ members.

Temporal balancing proactively migrates membership of coterie over time across nodes. To make it fast and cheap, CoMesh uses *zero-message exchange* protocols for selection—“zero-message” means that any smart device (in the fast path), without exchanging any messages, can calculate the current membership of any coterie. This mechanism uses zero foreground messages, relying instead on constant background

Table 3.2: **Key Terms used in this chapter, with examples for Figure 1.1b.**

Term	Description
Device	Any IoT device in the system. <i>Example:</i> lights, locks, shades, TVs, thermostats, fridges.
Command	A message sent to, and associated action executed by, one IoT device. <i>Example:</i> close shades.
Routine	A sequence of commands. Triggered by time, or a set of triggering clauses, or manually. <i>Example:</i> At 9pm, turn off all lights and lock all doors.
Smart Device (S)	A device that has (small) memory and computation capability. <i>Examples:</i> fridges, thermostats, TVs.
Simple Device (D)	An IoT Device that is not a Smart Device, like a sensor or actuator. <i>Examples:</i> lights, locks, shades.

bandwidth from the weakly-consistent membership protocol. This reduces both complexity and convergence time.

3.3 System Model

We assume a crash-recovery model: IoT devices may fail (crash) at any time, and then recover. We do not target Byzantine (malicious) faults. As is traditional in fault-tolerant literature, we assume that only up to f simultaneous crash failures occur in the system. The value of f can be calculated from deployment settings, e.g., in a smart building, by setting f to the number of devices in the largest power domain.

Table 3.2 summarizes key terms used in the paper. We assume a subset of devices (≥ 3) are smart devices, equipped with (small) memory and compute capability—other simple IoT devices may co-exist, capable only of receiving commands, responding to queries, but not executing computations.

Since IoT devices are typically fixed early in the operation of a smart building or smart farm, we assume that the locations of devices are known (we perform mobility experiments later). We assume an asynchronous system with unsynchronized clocks, where messages may be dropped.

Smart devices communicate via a mesh, i.e., a wireless ad-hoc network containing

device-to-device Wifi links. Meshes are known to be robust even when large fractions of nodes fail [93]. However, links can be lossy. An ad-hoc routing protocol [94] exists among devices in this mesh. We assume any smart device can send a command to any IoT device. Finally, CoMesh is layered atop a weakly-consistent membership service, e.g., Fuse [95], or Medley [96]. Such a membership protocol, running over an ad-hoc network, maintains a (nearly) full membership list (of all alive IoT device IDs) at each smart device, delivering updates only eventually. Like Medley [96], CoMesh assumes known device locations. Either precise (e.g., via a digital twin and map), or approximate device locations from localization algorithms work [97].

3.4 CoMesh’s Mechanism of Coterie

CoMesh balances the STA load across space and time by using *coterie*s—subgroups of smart devices, monitoring devices and managing routines. A coterie is configured with a (system-wide) parameter named k . To tolerate f system-wide failures, we set $k = (2f + 1)$. We present coterie internals (selection, election, quorum: Section 3.4.1), temporal migration (Section 3.4.2), failure response (Section 3.4.3), and locality (Section 3.4.4).

3.4.1 Coterie Operations: Selection, Leader, Quorum

CoMesh’s coterie selection needs to: (i) be load-balanced, and (ii) use a minimum number of messages. While some communication is unavoidable in any distributed system that aims to be fault-tolerant (e.g., pinging, membership, etc.), our key observation is that expensive distributed computing protocols (e.g., leader election, member selection, etc.) can be replaced with inexpensive “zero-message” variants.

Load-balanced Member Selection via Zero-message Exchange. A smart device with ID S discovers if it is present in a coterie for a target device with ID D by calculating a consistent hash: $Hash(S, D)$ [98], [99]. The alphabetically *lowest* k hashed IDs across the entire system are then the approved coterie members. This design is inspired by the idea of

cryptographic sortition [100]. Notice that, given a fixed hash function (e.g., a cryptographic hash like SHA-1), *any* arbitrary smart device can calculate the *entire* membership of *any* coterie by calculating this hash function on all known node IDs. With correct membership lists, this takes zero messages.

Leader Election. The coterie’s goal is to fault-tolerantly replicate the state of its assigned entity (device or routine). To make its operations efficient, each coterie elects a leader, selected as that non-faulty coterie member with the lowest hashed value $Hash(S, D)$. Along with using the zero-exchange mechanism for election, CoMesh ensures the leader is actually alive by running the (well-known) Bully election algorithm to quickly confirm the leader [101]. Bully election is attractive as it is fast in the common case: 1 round-trip time (RTT) for lowest-hashed coterie device (leader) to inform others in the coterie. Under failures Bully algorithm takes a worst case of 5 RTTs.

Quorum Agreement. Fast agreement in a coterie is done via quorums. In a coterie of size $k = 2f + 1$, anyone in a group (typically the leader) sends a message to all M members, and waits for acknowledgments from at least $(f + 1)$ members. Thus with f failures, at least one surviving coterie member knows about the latest decisions in the coterie, and can communicate it to future leaders.

3.4.2 Epoch-based Coterie Migration

While Section 3.4.1’s hashing provided *spatial* load balance, for temporal load balancing, CoMesh migrates each coterie membership periodically. Epoch lengths are fixed system-wide. (Different coterie can migrate at different times.)

First, we extend the hash function for the selection to include the epoch number E , making it $Hash(E, S, D)$. E is incremented by 1 on each epoch change. When an epoch expires, Algorithm 3.1 executes. Epoch changes are initiated asynchronously by the leader of the old coterie. When the new coterie leader acknowledges to the old coterie leader that it has received all the state (line 5), the old coterie leader can delete its old coterie state.

Algorithm 3.1 CoMesh Coterie Migration

Require: epoch E , current smart device S , smart device set $\mathcal{S}$, monitored device set $\mathcal{D}$, old coterie $\mathcal{M}_{old}$, old leader L_{old}

Ensure: $\mathcal{D}$'s coterie migration

- 1: $\mathcal{M}_{\mathcal{D}} \leftarrow \text{calc_members}(++E, \mathcal{D}, \mathcal{S})$ $\triangleright$ Using $\text{Hash}(E, S, D)$
 - 2: $L_{\mathcal{D}} \leftarrow \text{leader_election}(\mathcal{M}_{\mathcal{D}})$
 - 3: **if** $S == L_{old}$ **then**
 - 4: Replicate $state_{\mathcal{D}}$ to $L_{\mathcal{D}}$ $\triangleright$ (a)
 - 5: Receive and ack **CHANGEOVER** from $L_{\mathcal{D}}$ $\triangleright$ (e)
 - 6: Send **CHANGEOVER** to $\mathcal{M}_{old}$ $\triangleright$ (f)
 - 7: **else if** $S == L_{\mathcal{D}}$ **then**
 - 8: Receive $state_{\mathcal{D}}$ from L_{old} $\triangleright$ (b)
 - 9: Replicate $state_{\mathcal{D}}$ to quorum of $\mathcal{M}_{\mathcal{D}}$ $\triangleright$ (c)
 - 10: Send **CHANGEOVER** to L_{old} $\triangleright$ (d)
-

After line 6, an old coterie node can delete its old state for that coterie. During Algorithm 3.1's execution, no monitoring operations occur in the coterie. This means that old coterie members and the old leader cease all coterie operations at exactly the epoch changeover boundary. In practice, the epoch change is fast enough that this gap in monitoring is small, as shown by our experimental results. After the migration, all normal coterie operations resume.

3.4.3 Failure Response: Coterie Updates

If a device N_i fails, the decentralized failure detector layer informs all other smart devices of its failure. Because CoMesh's operations remain correct for group size $k \in [f + 1, 2f + 1]$ (a majority is still present), we trigger a coterie update only when f smart devices have failed (rather than on each failure)—this keeps the overhead of failure response low.

If the failed device N_i was a leader of a coterie, this triggers Section 3.4.1's election. This new leader needs to reconstruct the state from all the members in the coterie. This state is correct because of past quorums across at least $f + 1$ coterie members, so the new leader receives each past update from at least one non-faulty member. Once reconstructed, the leader sends this state to the coterie and waits for confirmation by a quorum ($f + 1$). If the new leader fails during reconstruction, the election is re-run, and this process repeats.

Algorithm 3.2 CoMesh LSH Majority Selection Rule

Require: Monitored device set $\mathcal{D}$, smart device set $\mathcal{S}$

Ensure: Selected smart device list R of size $f + 1$

```
1: Initialize  $N \leftarrow$  a random device in  $\mathcal{D}$ 
2:  $\{B_{\mathcal{S}_i}\} \leftarrow \{\text{LSH}(\mathcal{S}_i)\}$  for all  $\mathcal{S}_i \in \mathcal{S}$   $\triangleright$  buckets
3:  $B_N \leftarrow \text{LSH}(N)$ 
4:  $R \leftarrow \{\mathcal{S}_i\}$  where  $B_{\mathcal{S}_i} = B_N$   $\triangleright$  same bucket
5: if  $\text{size}(R) \leq f + 1$  then
6:    $B_{\text{neigh}} \leftarrow \text{LSH}(\text{neighbors of } N)$ 
7:    $R \leftarrow R \cup \text{smart devices in } B_{\text{neigh}}$ 
8: if  $\text{size}(R) > f + 1$  then Truncate  $R$  to size  $f + 1$ 
9: else Add random smart devices to  $R$  until  $\text{size}(R) = f + 1$ 
10: return  $R$ 
```

3.4.4 Grouping and Locality-Sensitive Hashing

Since CoMesh is running over an ad-hoc network, two issues arise: (i) the sheer number of devices and routines creates too many coterie, (ii) if coterie members are randomly selected (hashed), they will be spread out, making intra-coterie coordination slow and bandwidth-hungry.

1. Limiting the Number of Coterie. We use a genetic k-means clustering algorithm [102] to group the devices. Each resultant device group is assigned to one coterie. For routines, we pick a random device from its trigger set as its representative, and that device’s coterie becomes the routine’s coterie.

2. Locality-Sensitive Hashing. Selecting coterie randomly (Section 3.4.1) spreads the coterie members all over the network, slowing down both intra-group (election, quorum) and inter-leader operations. A challenge is to make coterie selection *local* while retaining spatial load-balancing benefits of hashing. We use physical device locations and adapt ideas from *locality-sensitive hashing (LSH)* [103]. In IoT, LSH has been used for flows [104] and outlier detection [105], but CoMesh is the first to adapt LSH for coordination protocols. Our LSH-based selection is compatible with zero message exchange protocols described earlier (Section 3.4.1). Classical LSH [103] hashes similar multi-dimensional vectors into the same “buckets” with high probability. Algorithm 3.2 details how CoMesh selects a device coterie’s $f+1$ majority using LSH.

Note that Algorithm 3.2 selects only the $(f+1)$ -member *majority* of a coterie locally (via LSH); the remaining f members are spread across the network by the base consistent hash (Section 3.4.1). This split is deliberate: keeping the $(f+1)$ -quorum local makes the common-case intra-coterie operations fast, while spreading the other f members ensures that a *correlated* failure of a single locality cannot wipe out an entire coterie. For example, an entire power domain may lose power at once; by our rule of thumb for setting f (Section 3.3), a power domain holds fewer than $f+1$ smart devices, so at least one nearby coterie member always survives such an outage.

At each epoch, we also desire a coterie to migrate only “nearby” so that STA traffic stays local, and state transfer to new coterie is fast.

The device’s location vector is d -dimensional and the *state vector* of a smart device is a $2d$ -dimensional vector that combines the device’s location with the average location of all devices it monitors, to preserve locality. For example, let $d = 2$ and (x_{S_i}, y_{S_i}) denote smart device S_i ’s coordinates. Let $\{(x_j, y_j)\}_{j=1}^m$ represent the coordinates of the m devices monitored by S_i during a given epoch. The averaged location of the monitored devices is: $x_{\text{ave}} = \frac{1}{m} \sum_{j=1}^m x_j$, $y_{\text{ave}} = \frac{1}{m} \sum_{j=1}^m y_j$. Thus, the *state vector* of smart device S_i at this epoch is: $(x_{S_i}, y_{S_i}, x_{\text{ave}}, y_{\text{ave}})$. For simple devices, we append the location vector $(x_1, \dots, x_d)$ so that all devices have the same dimensions for LSH.

CoMesh uses the LSH function $h_b = \lfloor \frac{\cdot + b}{r} \rfloor$, adapted from [103]. To reduce the probability of differing vectors being bucketed together, CoMesh includes two further optimizations: (i) concatenate hash functions, i.e., $H = (h_1, \dots, h_m)$; and (ii) use multiple (l) hashes. Thus CoMesh uses $\mathcal{H} = \{H_1, \dots, H_l\} = \{(h_{11}, \dots, h_{1m}), \dots, (h_{l1}, \dots, h_{lm})\}$. Table 3.3 summarizes the aforementioned LSH parameters.

Figures 3.4a and 3.4b show that increasing m over-clusters devices while raising l, r does the reverse. By default CoMesh sets $m = 2, l = 2, r = 4$. This has clear benefits across *system sizes*. Figure 3.5 evaluates 2D grid topologies ranging from 50 to 1000 devices, with the default parameters. The average quorum distance (Figure 3.5a) grows roughly linearly

Table 3.3: Summary of LSH parameters used in CoMesh.

Symbol	Definition	Default
a	Random Gaussian vector, same dimension as v [106]	—
b	Uniform scalar in $[0, r)$	—
r	Size of quantization interval	4
m	Number of concatenated hash functions	2
l	Number of independent hash groups	2

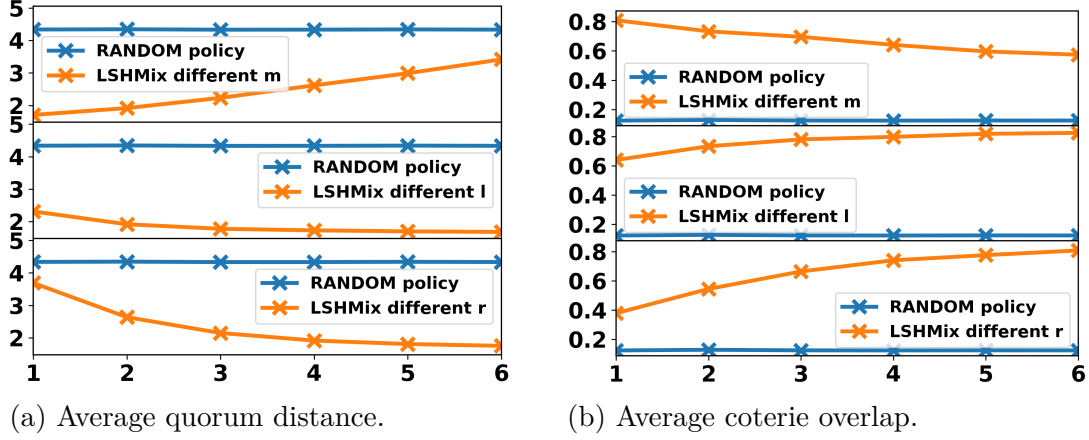


Figure 3.4: Average quorum distance and coterie overlap for different selection policies and parameters. $N = 500$, $\#smart\ devices = 200$.

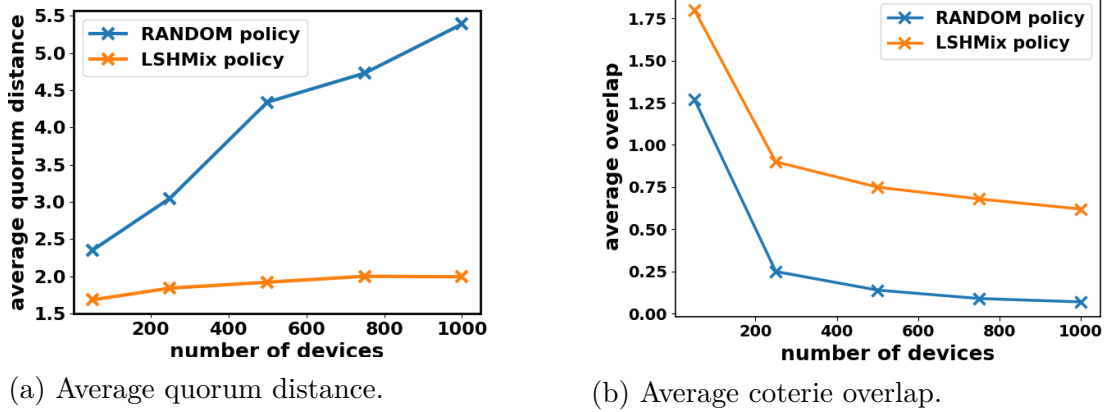


Figure 3.5: Average quorum distance and coterie overlap for different system sizes N . 2D grid, default $m = 2, l = 2, r = 4$, 40% smart devices.

with system size under random selection (from about 2.4 to 5.4), but stays small and nearly flat under LSHMix (about 1.7 to 2.0). The average coterie *overlap*—the number of members a coterie retains across two consecutive epochs, which lowers state-transfer cost at epoch changes—is consistently higher under LSHMix than random (Figure 3.5b).

Table 3.4: **Average number of coterie member candidates for different system sizes N (40% smart devices).**

# devices (N)	50	250	500	750	1000
# candidates	13.0	34.1	50.8	52.8	66.9

Although LSHMix’s overlap drops for large N (Figure 3.5b), it remains well above random’s. The drop is explained by Table 3.4: as the system grows, each coterie has more candidate members, so the probability of re-selecting the same device in consecutive epochs shrinks. One can also tune LSH parameters to reduce the candidate count without hurting quorum distance—e.g., at $N = 1000$ with $m = 3, l = 2, r = 4$, the average number of candidates is 21 and the average quorum distance is 2.4 (vs. 5.3 for random coterie selection).

3.5 Managing Devices and Routines in CoMesh

We now describe how CoMesh builds sense-trigger-actuate management of devices and routines atop Section 3.4’s coterie.

3.5.1 Device Management

In CoMesh, each device is assigned to its unique coterie, named the *device coterie*. This coterie is responsible for continually sensing the device, i.e., tracking the up/down status of the device, and the latest state of the device (e.g., temperature readings for a thermostat). The leader of the device’s coterie (henceforth called the *device leader*) keeps track of the device’s latest status. The device leader periodically “polls” the device for its state. (Alternately, if the device supports it, it may receive “pushes” of state change from the device itself.) If the device does not respond within 2 RTTs, the device is presumed failed, and the device leader notifies all smart devices in the topology via a flooded message. If the detected state of the device D (received in the ack) differs from its latest recorded state at the device leader, the device leader sends this new state to all leaders of the routines

which contain D in their triggering clauses.

3.5.2 Routine Management

Recall that a routine consists of an arbitrary boolean trigger clause (on multiple devices and their states) and a set of commands to execute after the routine is triggered. Call the set of devices in the former the *trigger set* and the latter devices as the *command set*.

(These sets may or may not overlap.)

CoMesh assigns each routine to its unique coterie, named the *routine coterie*. This coterie is tasked with triggering, and actuation (execution) of the routine. The sensing is done by device coterie (Section 3.5.1). The leader of the routine’s coterie (henceforth the *routine leader*) communicates with the device leaders of its trigger set in normal times (when the routine is not triggered). Once the routine is triggered, the routine leader communicates with the device leaders of the command set.

Recall that CoMesh needs to prevent conflicting routines (actuating intersecting device sets) from executing concurrently. CoMesh uses “virtual” locking of devices (inspired by [90]). Before a routine starts, it needs to acquire locks for all devices it might affect, and when it ends it releases all locks. Thus a routine may be in one of the following states (maintained at its coterie): (i) NOT_TRIGGERED, (ii) ACQUIRING_LOCKS, (iii) EXECUTING, and (iv) RELEASING_LOCKS.

The routine leader contacts the device leaders of each device in the routine’s trigger set. Whenever a device leader sends an updated state to the routine leader (Section 3.5.1), the latter checks if this state change satisfies the trigger clause for the routine.

Once a routine is triggered, its routine leader changes its state from NOT_TRIGGERED, to ACQUIRING_LOCKS, and replicates the state to all coterie members. The routine leader awaits $(f + 1)$ responses, including itself, then acquires (virtual) locks to all devices in the routine’s command set, and executes the commands. Finally, when the routine is finished the routine leader releases all locks to the command set’s device leaders.

The lock for a given device is only maintained in that device coterie (rather than the routine coterie)—this allows multiple routines to compete for locks. For a routine to acquire a device’s lock, the routine leader contacts the device leader. (If the routine leader is unaware of the current device leader, it contacts all device coterie members, known via Section 3.4.1’s zero-message exchange protocol, to find the leader.) The message is retried until a response is returned or the coterie member is detected as failed. In that case, a new member is recruited, synced to the current state, and queried anew.

The device leader maintains a *wait list* of routines requesting a lock, and grants only one lock at a time (in FIFO order). When a routine leader releases a lock (at `RELEASING_LOCKS` stage), the device leader dequeues the next entry from the wait list, sends a request to all its device coterie members, and waits for $(f + 1)$ acks, before giving the lock to this requesting routine leader. When a routine leader has acquired all locks for its actuated devices, it changes the routine’s state to `EXECUTING`, communicates this to all routine coterie members, waits for $(f + 1)$ acks, and then starts issuing the commands for the routine. Each command is routed from the routine leader directly to the device.

CoMesh can acquire locks either serially or parallelly:

- 1. Sequential Lock-Acquiring (SLA) Strategy.** The routine leader acquires locks sequentially. If locks are acquired in increasing order of device ID, this prevents deadlocks (no cycles) among routines with conflicting device sets. The SLA strategy can be slow when there are no or few conflicts.
- 2. Optimistic Lock-Acquiring (OLA) Strategy.** This approach attempts to acquire all locks for its (desired) devices simultaneously. If any lock request fails, the routine leader releases all locks it acquired, and retries all again. To prevent message implosion, retries occur after a backoff timeout. OLA can be slow when routines’ command sets overlap.

3.6 Formal Analysis

We formally analyze CoMesh’s properties. For the reader who wishes to skip this section, we summarize our findings:

- *Inheritance*: When a coterie changes, the state held by the old leader and new leader are identical.
- *Safety*: No two routines that touch an overlapping set of devices, are allowed to execute simultaneously.
- *Liveness*: No routines deadlock.
- *Progress*: Every routine makes progress.
- We also calculate the (probabilistic) availability of the system when more than f devices fail.

3.6.1 Inheritance

Definition 3.1. *Inheritance means that given a coterie, the old and new versions of the coterie—before and after epoch change, or after leader failure—maintain identical state.*

For a device coterie, the state transferred to the new leader includes: the device’s current availability, and the lock queue. For a routine coterie, the new leader should receive: the routine state, and the IDs of devices already locked and released.

Lemma 3.1. *Inheritance is guaranteed if at most one node fails during state transfer.*

Proof. We consider two cases that trigger state transfer:

Case 1—Leader Failure: Upon coterie leader L_o ’s failure, the new leader L_n requests and receives from all coterie members, their local coterie state. L_n includes a(ny) state entry e in the recreated state if and only if there is at least 1 received state that contains e . For the device coterie, every routine request in its lock queue is tagged with a sequence

number denoting when it reached D_k 's coterie leader— L_n uses these to sort the queue.

Since each routine entry e in the lock queue was replicated by at least a quorum of coterie members before the old leader L_o 's failure, L_n is guaranteed to receive the exact same set (and sequence) of routine requests as L_o .

Case 2—Epoch Change: New leader L_n receives old leader L_o 's state. The single failure during state transfer may be:

Case 2a— L_o fails: L_n detects L_o 's failure, and requests states from all old coterie members. This becomes Case 1.

Case 2b— L_n fails: A new leader L'_n is elected (as per protocol) and requests state from L_o .

Case 2c—member of old coterie fails: This does not affect state transfer between L_n and L_o .

Case 2d—member of new coterie fails: A new node replaces the failed one, and it gets the new state from L_n . □

Lemma 3.2. *Inheritance is guaranteed if there are at most f node failures during state transfer.*

Proof. If either there are no epoch changes (Lemma 3.1: Case 1), or failures happen only among non-leader members during epoch change (Lemma 3.1 Cases: 2c, 2d), each routine entry e is maintained by at least $f + 1 - f = 1$ alive nodes. Thus, Lemma 3.1: Case 1 will recreate state correctly.

Upon epoch change, if old leader L_o fails before new leader L_n gets state, L_n (Lemma 3.1: Case 2a) reconstructs the same state as L_o since each state entry e is maintained by at least $f + 1 - f = 1$ alive old coterie nodes. Repeated L_n failures cause the new leader to re-attempt state reconstruction.

If L_n fails before L_o , Lemma 3.1: Case 2b applies. □

3.6.2 Safety

Theorem 3.1. [**Safety**] *No two routines that conflict in devices (i.e., their touched device sets are not disjoint) can execute simultaneously, under at most one node failure.*

Proof. Given routines R_1, R_2 that conflict in devices, we analyze OLA and SLA (Section 3.5.2) separately.

SLA: Let D_k be that shared device with the minimum ID . If D_k 's coterie leader L_k does not fail, by definition, at most one routine can reach quorum and therefore acquire the device lock. During leader transition, all new requests are queued at the new leader L_n . Hence, we only discuss outstanding requests that have been approved when L_k fails:

Case 1: If L_k fails before it has replicated the request from R_1 to a quorum of coterie nodes, R_1 's coterie leader resends R_1 's request after a timeout of not receiving an acknowledgment.

Case 2: If L_k fails after replicating R_1 's request to a quorum in the coterie, new leader L_n will receive R_1 from this quorum.

Case 3: If L_k fails after R_1 's request reaches the top of the lock queue, two sub-cases occur: (3a) Request approved by a quorum of nodes: new leader L_n marks R_1 as the approved routine and notifies R_1 's coterie leader. (3b) Request has not yet reached quorum: L_n adds R_1 to the lock queue.

Case 4: If L_k fails after it has sent a Locked message to R_1 's coterie, L_n : 1) re-constitutes the lock queue from surviving nodes, and 2) (re)notifies R_1 's coterie leader of its approval.

OLA: For a device D_k touched by R_1 & R_2 , if D_k 's coterie leader L_k does not fail, only one of R_1, R_2 can reach quorum and be approved to lock D_k . If L_k fails, four cases arise:

Case 1: If L_k fails before the quorum's pre-lock replication ack, new leader L_n will have no record of it. R_1 's coterie leader will resend the request after a timeout.

Case 2: If L_k fails after quorum replication but before hearing back from R_1 's leader (to lock for execution or release due to failed pre-lock), L_n will know R_1 has pre-locked D_k from its group members, and resend pre-lock information back to R_1 .

Case 3: If L_k fails after hearing from R_1 (lock acquisition/releasing), but before getting a quorum acknowledgment, R_1 's coterie leader will resend the request after a timeout.

Case 4: If L_k fails after it has locked D_k and notifies R_1 's coterie L_n : 1) re-constitutes the state from surviving nodes, and 2) (re)notifies R_1 's coterie leader of its approval. $\square$

3.6.3 Liveness

Theorem 3.2. [No Deadlocks] *No routines deadlock.*

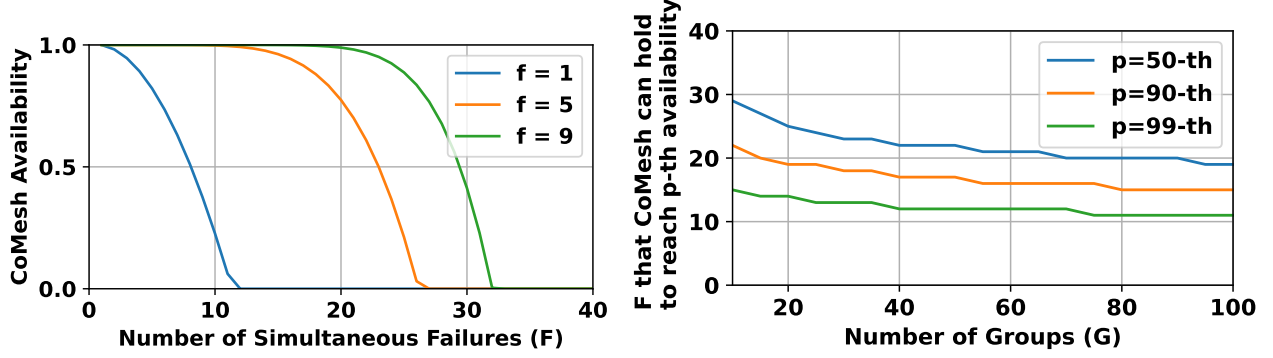
Proof. Each of SLA and OLA violate separate necessary conditions for deadlocks. SLA avoids “wait for” cycles among routines, since routines only wait for higher-*ID* devices than they have already locked. OLA violates the “hold and wait” condition since routines do not wait (and instead abort) for any already-locked devices. Leader change and failure do not affect device locks held by CoMesh. Meanwhile Lemmas 3.1 and 3.2 guarantee inheritance. Thus lock assignment is not influenced by leader changes and failures, i.e., there is no deadlock. $\square$

The following theorem can be applied inductively.

Theorem 3.3. [Progress/Liveness] *Consider a set of executing routines $\mathcal{R}$ still waiting to acquire 1 or more locks, and a set of devices $\mathcal{D}$ they are waiting for. If no further routines arrive into $\mathcal{R}$, and SLA is used for locking, then: at least one routine from the set $\mathcal{R}$ will make progress (i.e., get access to one more device among the devices that it desires).*

Proof. Consider the highest-*ID* device D in set $\mathcal{D}$, and the routine R that currently holds the lock on D . R cannot be waiting for any further locks (otherwise D would not have the highest-*ID* in $\mathcal{D}$). When R eventually completes, the routine at the head of D 's queue will get access to D . $\square$

For OLA, an adversary may always be able to prevent two competing routines from making progress. However, in practice, we observed that OLA makes fast progress. We use SLA as the default in the CoMesh implementation.



(a) Availability under different number of simultaneous failures.

(b) Impact of different number of groups in the system ($f = 5$).

Figure 3.6: **Probability $P(F)$ of CoMesh running correctly under different parameters.** Default $S = 100$, $G = 30$, $k = 2f + 1$.

3.6.4 System Availability

Given S smart devices, G coterie each containing k members, and F = total actual node failures during a run, we calculate the *availability* $P(F)$ = probability that CoMesh still works correctly under F total failures, i.e., all coterie contain $\geq (f + 1)$ non-faulty nodes.

1. When $F \leq f$, $P(F) = 100\%$.
2. When $F > f$, the *availability* is:

$$\begin{aligned}
 P(F) &= P(\text{at most } f \text{ failures across all coterie}) \\
 &\geq 1 - \sum_{i=1}^G P(> f \text{ failures in one coterie}) \\
 &= 1 - G \cdot \sum_{t=f+1}^{\min(k, F)} \frac{\binom{k}{t} \binom{S-k}{F-t}}{\binom{S}{F}}.
 \end{aligned}$$

t counts the number of failures in a coterie, ranging from $(f + 1)$ to $\min(k, F)$. The denominator counts all possible F failures in the system. The numerator's (i) left term counts all ways of selecting t failures in a coterie of size k , and (ii) right term shows the selection of the remaining $(F - t)$ failures.

Figure 3.6 plots this equation. Beyond $F > f$, Figure 3.6a shows CoMesh provides 50% of availability when F grows as high as $2 \times f$. Figure 3.6b shows that to reach a specific

percentile of availability p , CoMesh’s tolerated F drops very slowly, indicating scalability with the number of groups.

3.7 Experimental Evaluation

We present: (i) Raspberry Pi Lab Deployments (Section 3.7.1) driven by mini-maps of a smart building and a smart farm, (ii) Large-scale Simulations (Section 3.7.2), & (iii) Map-based Simulations with real Routine workloads (Section 3.7.3).

3.7.1 Raspberry Pi Deployments

We implemented CoMesh in Raspberry Pi (RPi) 4, via 12.9K lines of Java code. Our deployments use commercial RPi 4 model B, with 2GB of LPDDR4 RAM, Broadcom BCM2711 quad-core CPU of 1.5GHz Cortex-A72 cores, and communicating via WiFi. To implement the central approach, we use an additional 17th Pi device as the central hub. To create the mesh topology, we attenuate RPi’s strong signal by consistently wrapping each device in aluminum foil to lower transmission power to 15dBm [90]. We use OLSR routing [94] as it is standard in Pi 4s. For membership, we use Medley [96].

CoMesh is configured to use coterie with expected $k = 3$ smart devices per coterie, and a device coterie monitors its devices every second. The default locking is SLA locking (Section 3.5.2). To avoid cold start biases, we start measurements 10 s after CoMesh is initialized. Each data point involves at least 5 trial runs with different seeds. Epoch length is 60 s.

Routine workload. Multiple routines may be triggered independently and simultaneously; the same routine is never re-triggered within 10 s of its own previous invocation. For each routine, its number of triggering (resp. actuated) devices is drawn from a Gaussian distribution ranging from a low of 2 (resp. 1) to a high of $N/4$, with a mean of 2 and a standard deviation of 1; triggering and actuated devices are then chosen uniformly at random. We inject new device-state events uniformly every $[0.5, 1]$ s.

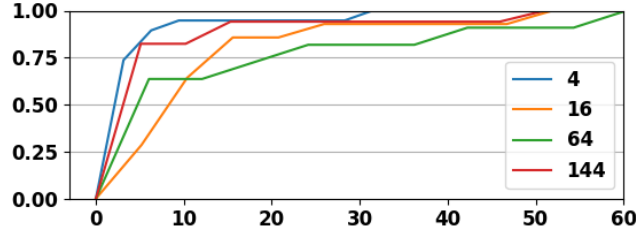


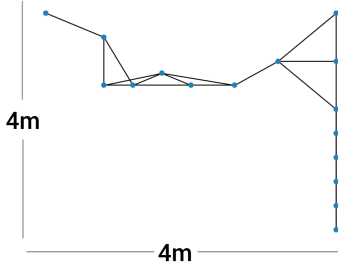
Figure 3.7: **Deployment: CDF of routine inter-arrival times (seconds) for different device counts.** $N = 4, 16, 64, 144$.



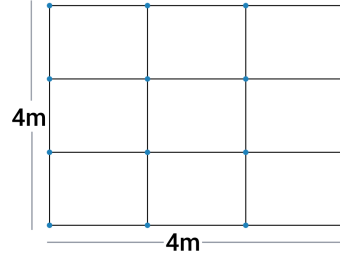
(a) Mini Smart Building (MOB).
From Figure 1.1b.



(b) Mini Smart Farm (MSF).
From Figure 1.1c.



(c) Mini Smart Building (MOB).
From Figure 1.1b.



(d) Mini Smart Farm (MSF).
From Figure 1.1c.

Figure 3.8: **Deployment: Raspberry Pi Topologies.** *From Figures 1.1b and 1.1c.*

Figure 3.7 shows the CDF of the resulting routine inter-arrival times for each of our device counts ($N = 4, 16, 64, 144$).

We implement the two Pi topologies shown in Figure 3.8, based on Figures 1.1b and 1.1c. They are: (i) a *Mini Smart Building* topology (*MOB* in plots), based on the L-shaped map of the Smart Building from Figure 1.1b; and (ii) a *Mini Smart Farm topology*

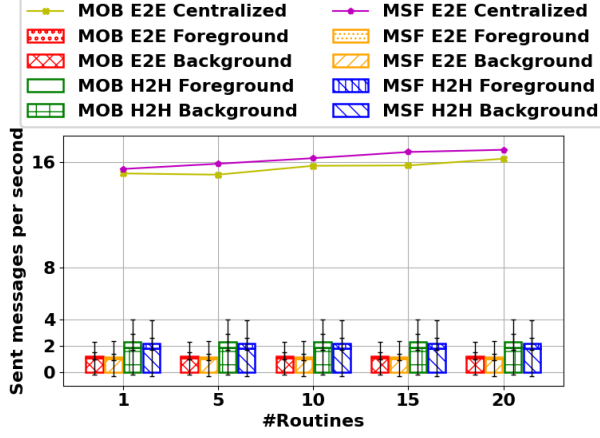


Figure 3.9: **Deployment: Centralized (lines) vs. CoMesh (bars): Bandwidth.** *MOB = Mini Smart Building, MSF = Mini Smart Farm.*

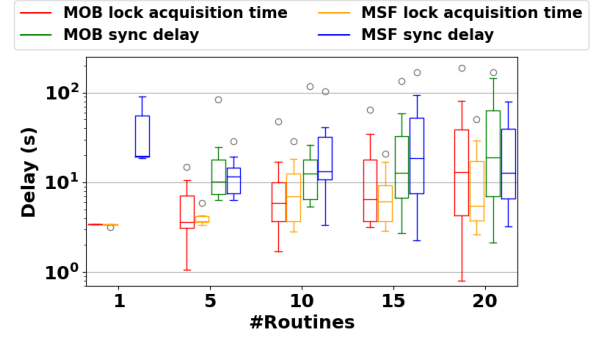


Figure 3.10: **Deployment: Lock Acquisition Time and Sync Delay.** *MOB = Mini Smart Building, MSF = Mini Smart Farm.*

(*MSF* in plots) based on Figure 1.1c, modeled via a 4×4 2D Grid. Each covers an area of 4 meters $\times$ 4 meters. Note the *MOB* topology’s diameter (in hops) is higher than *MSF*’s.

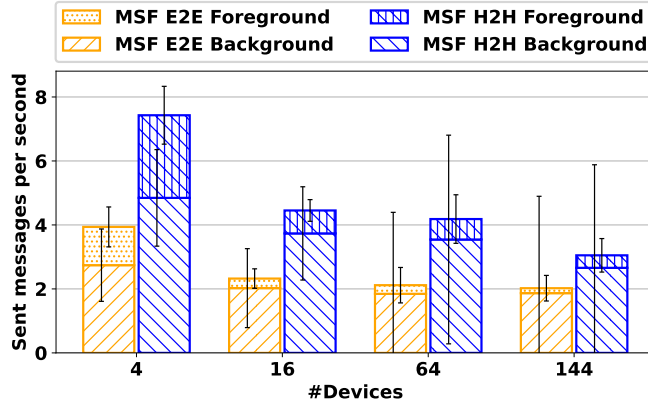


Figure 3.11: **Deployment: Scalability of Per-(Virtual) Device Bandwidth.** *MSF = Mini Smart Farm.* Error bars—left: background, right: foreground.

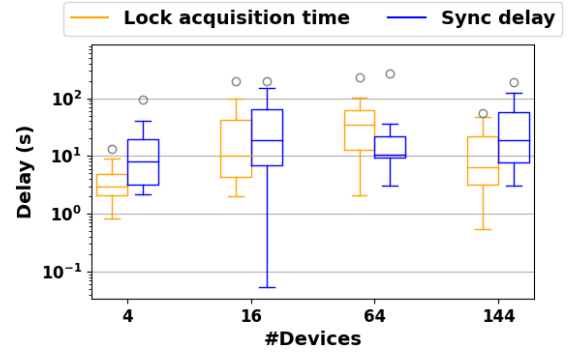


Figure 3.12: **Deployment: Scalability of Lock Acquisition Time and Sync Delay.** *MSF = Mini Smart Farm. 20 Routines.*

CoMesh Consumes Less Bandwidth than a Central Hub. Figure 3.9 shows decentralized CoMesh’s bandwidth per smart device, both end-to-end (E2E) messages, and ad-hoc routed (hop-to-hop/H2H) messages. For fair comparison, we model a central hub over a mesh network (Section 3.1) by instantiating CoMesh with $k = 1$ and one coterie, i.e., a single centralized device responsible for all routines’ management. We observe that: a)

CoMesh reduces bandwidth by an order of magnitude (over $10\times$) compared to the centralized approach. b) Both end-to-end bandwidth and hop-to-hop bandwidth are scalable and remain flat, as the number of routines grows. c) Bandwidth of foreground operations (e.g., locking) remains small compared to background bandwidth (e.g., device monitoring).

CoMesh Scales with Routine Load. Figure 3.10 shows two per-routine metrics. The *Lock Acquisition time* is the average time for a routine to acquire a device lock after it becomes available. We observe the Lock Acquisition time varies between 3.41 s - 13.02 s. As the routine count increases, the median remains stable for Mini Smart Farm (MSF), and rises slightly for Mini Smart Building (MOB) due to longer diameter and route competition. The second metric is *Sync delay*, the handover time from a finishing routine to the next already-waiting (conflicting) routine starting its first command. We observe that with more routines, Sync Delay stays stable for both MOB and MSF, with medians between 10.23 s - 20.833 s.

CoMesh Scales with Number of Devices. We scale up by overloading each (of the 16) Pi devices as multiple “virtual devices” (henceforth “nodes”). The rest of CoMesh’s operation remained the same. We inject 20 routines over 10 mins in the MSF topology. Figure 3.11 shows that up to 144 devices both end-to-end (E2E) and hop-wise (H2H) bandwidth *drop* with scale, and plateau. Background bandwidth (lower stack) plateaus quickly as the number of node coterie increases at the same rate as nodes, thus spreading load evenly. Foreground bandwidth drops much quicker as the fixed 20 routine coterie (20) are spread over an increasing number of nodes.

Figure 3.12 shows that both startup times for a routine—Lock Acquisition time and Sync delay—rise slowly but then plateau with scale. Essentially, with more nodes, inter-routine conflicts become rarer. Median Lock Acquisition time stays in 2.97 s - 34.71 s, and Sync Delay median is in 8.01 s - 19.04 s.

CoMesh Internal Operations are Fast. Table 3.5 shows that in MSF topology on the

Table 3.5: **Coterie Latencies in Mini Smart Farm topology.**

Coterie delay (seconds)	median	mean	q90
Quorum reply	1.0805	1.4021	1.4962
Leader election	1.1675	1.4913	1.9249
State transfer	2.0750	2.8468	3.1390

Table 3.6: **Default parameter values for simulation.**

Parameter	Default value
# total devices (N)	250
# failures for completeness (f)	2
coterie size (k)	5
# smart devices (S)	100 (40% of all devices)
epoch length	200 time units
max routine length	5 commands
avg devices per routine	5
# devices managed by a Coterie	25
node bandwidth cap	625 KBps/5Mbps
coterie selection policy	LSH-random mix
device cluster policy	locality
leader election policy	LSH smallest hash
default LSH parameters	$m = 2, l = 2, r = 4$

Pis, CoMesh’s common operations are fast and take a median of 1.08 s for quorum, 1.17 s for election, and 2.08 s for state transfer during epoch change.

3.7.2 Large-Scale Simulation

Our second set of evaluations measures CoMesh at larger scales with 100s of devices, via simulation. Since existing simulators like NS3 [107] have known scalability issues beyond a couple of hundred nodes, we wrote a custom simulator that is scalable but models bandwidth on network hops. We measure both: i) user-facing metrics, e.g., routine startup latency, load balance, etc., and ii) microbenchmarks of coterie operations.

Our simulator is written in Java and comprises 17.2K lines of code. Table 3.6 shows default parameter settings. 1 time unit in simulation is equivalent to 1 ms of real time. It models network contention by scaling down network bandwidth based on density: bandwidth from device A to device B is the default node bandwidth divided by A ’s 1-hop

neighbor count. The default node bandwidth (Table 3.6) was set based on reports in user complaints at online IoT forums [108]. Latency on each hop is fixed at 1 time unit (we later vary hop latency). Ad-hoc routing uses Dijkstra’s algorithm. Varying the fraction of smart devices from 5% to 40% had no discernible effect on coterie latencies—so we use 40% smart devices by default.

We use three connected network topologies: (1) 3D grid, (2) Random, and (3) Clustered. We use LSHMix (Section 3.4.4).

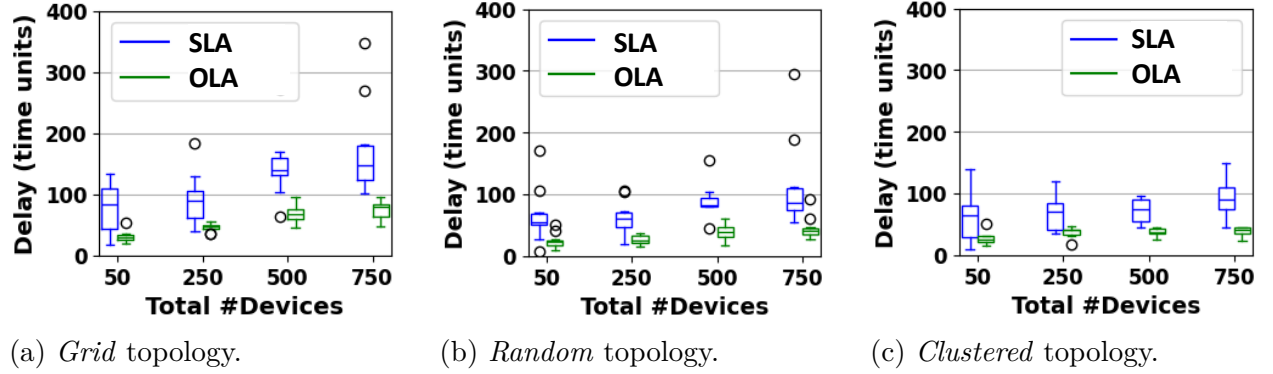


Figure 3.13: **Client Delay: Sequential (SLA) vs. Optimistic (OLA) Lock-Acquiring Strategy, in three topologies.** 1 time unit is equivalent to 1 ms of real time.

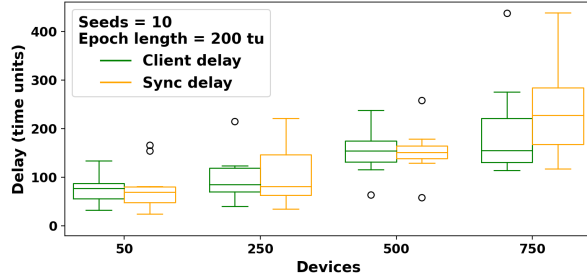


Figure 3.14: **Client Delay and Sync Delay** when 40% of the smart devices in a grid topology experience churn (either a join or a failure).

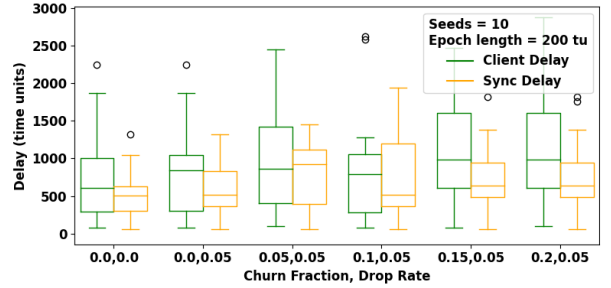


Figure 3.15: **Client and Sync Delays of routines under failure vs Churn fraction and packet Drop rate (80% smart devices).**

Client Delay. Suppose there are no routines active. Then, if one routine’s trigger conditions are activated, the time until executing its first command is the *client delay*—this measures a routine’s cold-start time. Client delay includes time for the routine coterie to

detect (and replicate) the trigger, lock all of its actuated devices, and start execution. In Figure 3.13, we observe that CoMesh’s client delay scales (increases gently) with device count, regardless of topology or locking strategy. As expected, optimistic locking (OLA) is faster than sequential locking (SLA) because client delay excludes conflicting routines. Among the 3 topologies, grid has higher client delay than random and clustered (which are similar), because the latter two have local density variations that form node “clusters” in the network which tend to speed up communication. Even though OLA is faster, we could prove liveness only for SLA [109]. Users should employ OLA only in low-packet-drop networks, while SLA is robust, general-purpose, and still scalable. Scale goes up to 750 devices, similar to our dense smart building topology from Section 3.7.3.

Churn Effect on Client/Sync Delays. Churn refers to back-to-back smart device arrivals, departures, and failures. Even if membership lists are updated quickly (by underlying membership protocol Medley [96]), coterie operations still incur overhead. Figure 3.14 shows that when 40% of the smart devices are churned in a grid topology (SLA locking), client delay increases only 25%, as some routines face the overhead of failover (election and state transfer). To measure Section 3.7.1’s *sync delay*, we triggered two back-to-back instances of the identical routine. Figure 3.14 shows sync delay rises slowly—raising device count by $15\times$ from 50 to 750 causes $< 4\times$ increase in sync delay (and client delay). Note that client delay does *not* subsume sync delay. Sync delay excludes triggering of routine and associated coterie quorum. Client delay excludes lock release latency and associated quorums.

Effect of Stale Membership due to Drops and Churn. *False detections* by the underlying membership may arise from dropped messages that make a failed device indistinguishable from a lossy one. This leads to inconsistent membership, and may slow CoMesh’s zero-message exchange mechanisms. Yet, with 1000 devices and 80% of them smart, Figure 3.15 shows that raising packet drop rate (on all hops) from 0%-5%, along with churn fraction (% smart devices failed during a run) keeps both client delay and sync

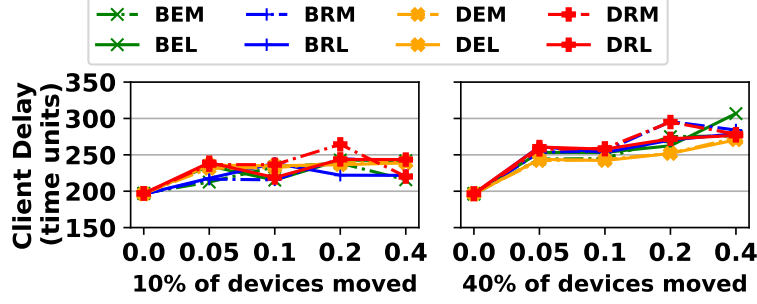


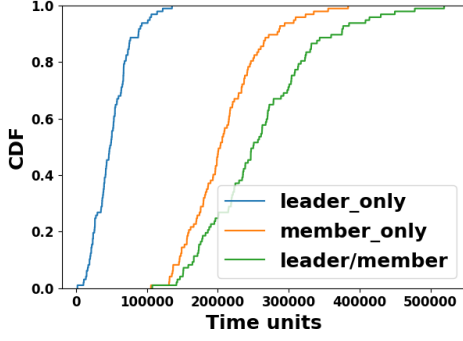
Figure 3.16: **Mobility: Client delay vs. relative distance moved.**

delay stable. Our churn and packet loss rates are much higher than in commercial deployments, which target 99.9% uptime for devices [110].

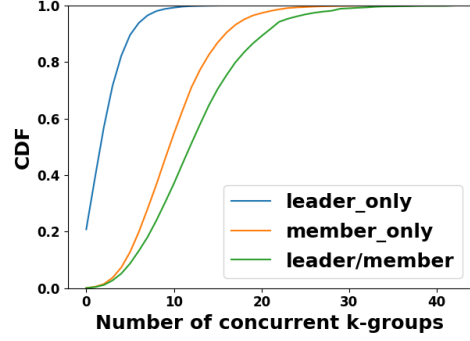
Device Mobility. Figure 3.16 shows the effect of device mobility, evaluated at different points of CoMesh operation. With 250 devices organized in a $5 \times 5 \times 10$ grid, we instantly move a fraction of the devices, each randomly by distance in the interval $[0, X]$. After the instant move, routing tables are updated (for correct routing), but *we do not update the soft state of CoMesh and keep it stale*. Figure 3.16 shows 8 combinations created by injecting the mobility either at the start of or during a CoMesh operation (B,D), during epoch change or routine triggering (E,R), and excluding or including routine leader (M,L). For space, we only plot client delay (sync delay showed similar behavior). We observe that with up to 10% mobility CoMesh raises client delay by less than 25%. The degradation gets worse at higher mobility (40%), but CoMesh stays safe. We conclude that CoMesh can tolerate moderate mobility.

Load Distribution. Figure 3.17 measures two kinds of load distribution over a run: (1) *Temporal load* is the time that a smart device serves in a particular role. (2) *Spatial load* is the number of concurrent coterie a smart device is present in. The plots show LSH’s clustering of coterie closer to target devices prolongs tail load vs. random (Figures 3.17a and 3.17b) due to repeated elections. Yet LSH’s median temporal load for member and leader/member are similar to random’s (Figure 3.17c), showing LSH balances load well.

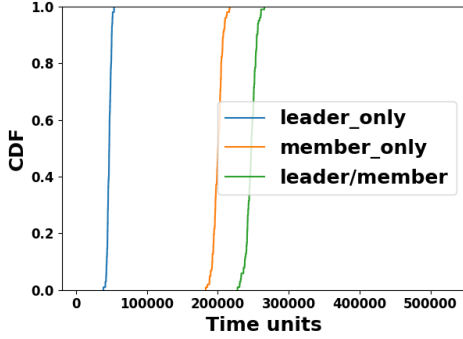
Coterie Operations. Figure 3.18 measures coterie internal performance (default parameters: Table 3.6). All coterie operations—election, quorum, state transfer—scale as



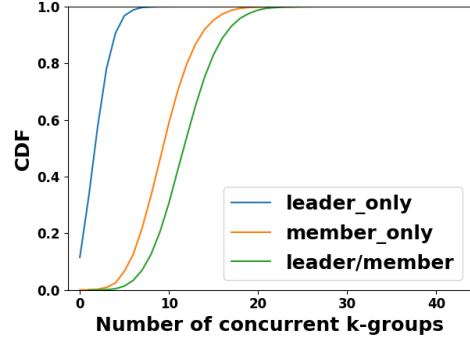
(a) *Temporal Load, LSH.*



(b) *Spatial Load, LSH.*



(c) *Temporal Load, Random.*



(d) *Spatial Load, Random.*

Figure 3.17: **CoMesh’s Load Distribution in Time & Space: LSH/locality vs. Random (member & leader selection).** *Grid topology.*

coterie size is increased from 3 ($f = 1$) to 11 ($f = 5$). Quorums are scalable because requests and replies are sent in parallel. Leader election delays are scalable as the Bully algorithm is fast in the common case. State transfer delay is a function of the amount of state, but does not depend on coterie size k .

Figure 3.19 confirms our hypothesis that locality-awareness (LSH of (Section 3.4.4)) decreases operational delays of a coterie. The improvement from LSH is more prominent for election and quorum due to the localized communication (state transfer is dominated by the old leader to new leader bandwidth).

Figure 3.20 shows a coterie’s operations are not affected by churn among smart devices, except beyond 60% churn. Essentially, requiring only a quorum imbues churn-tolerance. When f members fail, the coterie’s leader recruits replacements to tolerate future failures (Section 3.4.3).

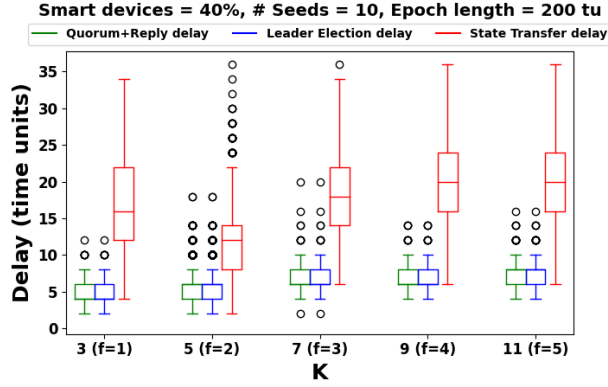


Figure 3.18: Delays of operations within a Coterie for different k .

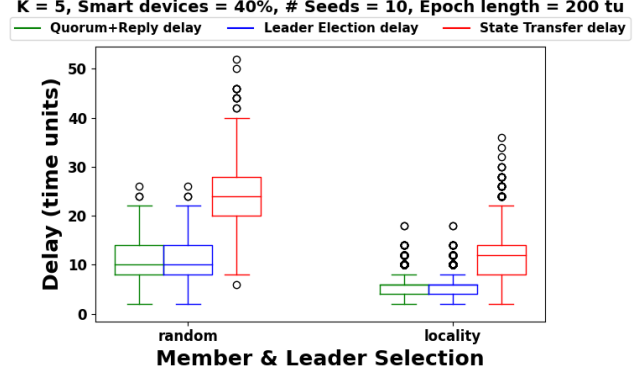


Figure 3.19: Delays of operations within a Coterie for different policies of entity clustering, member selection, and leader election.

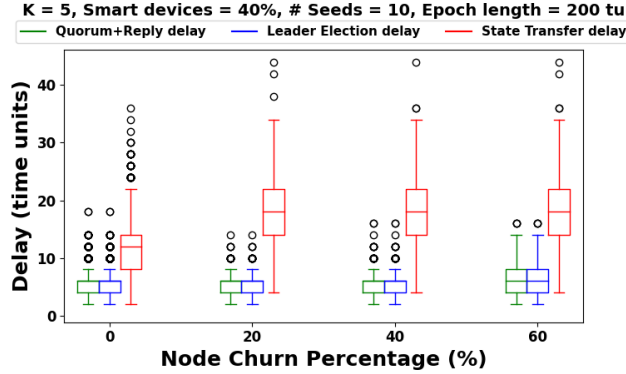
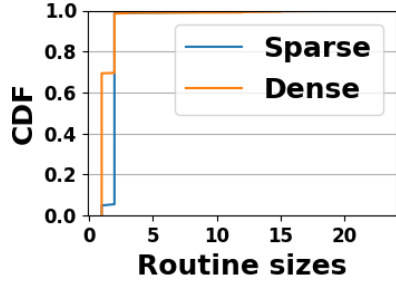


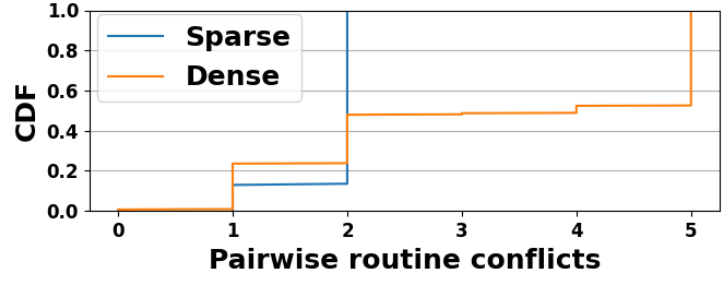
Figure 3.20: Coterie Internal Operations vs. Smart Device Churn.

3.7.3 Map-Driven Simulation with Real Workloads

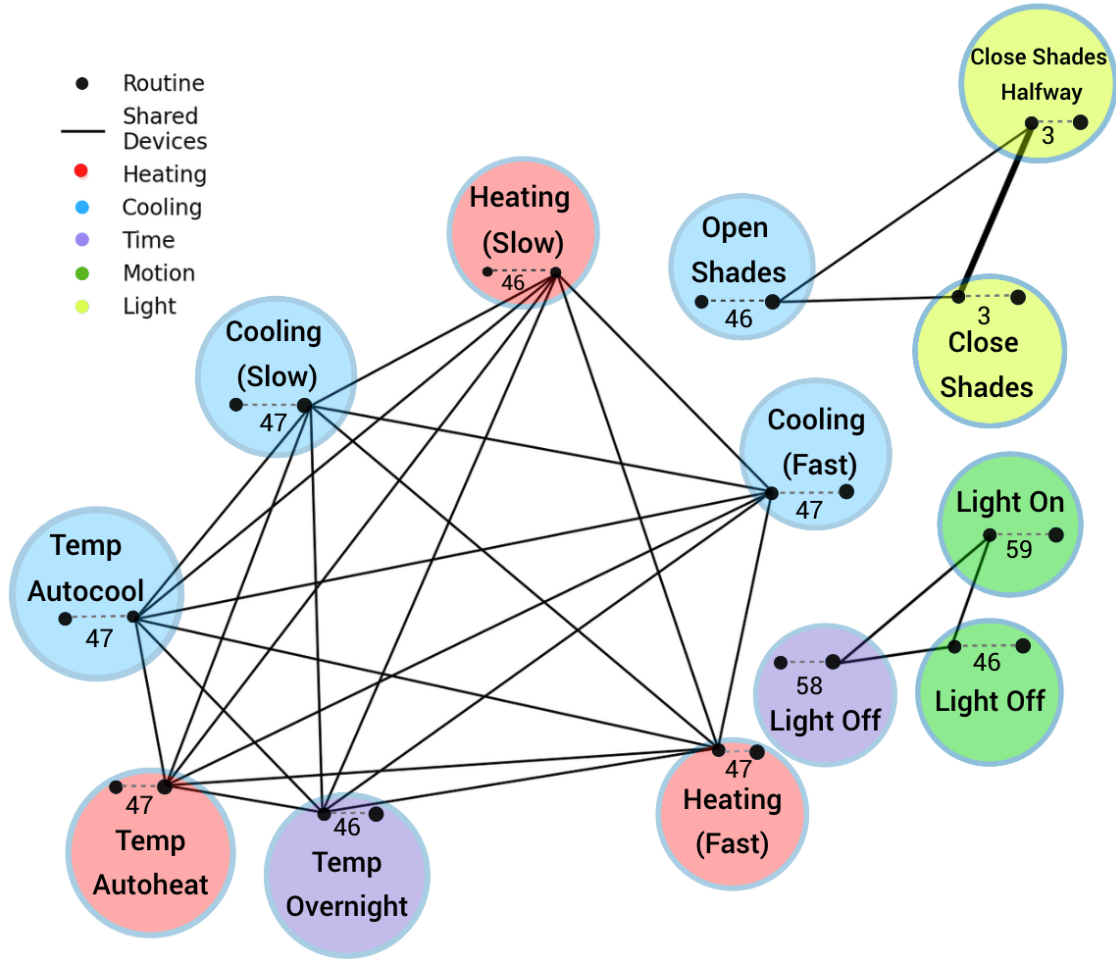
We perform a large simulation using a real building map and a realistic routine workload mix. The Smart Building (Figure 1.1b) is a “living laboratory” sized 250 ft $\times$ 362 ft, with sensors and actuators in offices, lecture halls, and conference rooms. Map-driven simulation allows us to flexibly vary device densities, transmission radii, and routine sets. Our simulator (Section 3.7.2) includes this map with two topologies: (i) a *dense* topology with 708 devices, and (ii) a *sparse* topology with 285 devices. Dense topology has 4.06 ± 2.31 devices per room & corridor. Sparse topology has 3.43 ± 1.79 . 40% of devices are smart. Sparse has motion sensors and lights, while dense also has thermostats, light sensors, shades, and microphones.



(a) CDF of how many devices a routine actuates.



(b) CDF of the number of conflicting actuated devices between routine pairs.



(c) Routine conflicts for the Sparse (bottom right 3 classes) and Dense topologies. Circle = routine class; Color = trigger condition; Count = number of routines; Thicker edges = more conflicting devices.

Figure 3.21: **Workload in Smart Building Map-Driven Simulations.** *Dense topology: all 545 routines. Sparse topology: 163 routines.*

We created a routine workload mix based on devices and frequently observed actions from this building. Figure 3.21(a,b) show a CDF of device count actuated by a routine,

and pairwise routine conflicts. Figure 3.21c shows major routine classes (only 2 representatives shown out of many), and typical conflicts. For concreteness, we describe a few sample routine classes. When motion is undetected for at least an hour and a room’s light is on, a routine turns it off; conversely, when motion is detected, another routine turns the room’s lights on. Other routines turn off room and corridor lights overnight. These per-room routines conflict with building-wide routines—e.g., ones that turn off all lights everywhere between 2–6 AM—and similar local–global conflicting pairs arise for thermostats and shades. All routines are triggered by sensors. The dense (respectively sparse) topology has 545 active routines (resp. 163), containing 1–23 commands with mean 1.47 and median 1 (resp. for sparse: 1–12, median: 2, mean: 2.07). We calibrated the simulator so that 1 time unit in the simulation is equivalent to 1 ms of real time. Based on our measurements of inter-Raspberry Pi latencies, we set the hop latency realistically to $(1.4 \times e^{(0.03 \cdot \text{distance})})$ ms.

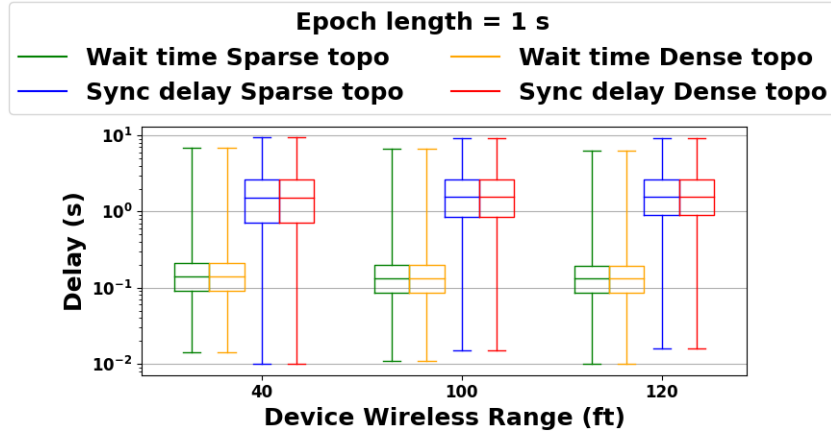


Figure 3.22: **Smart Building (Map-driven) Simulation (Figure 1.1b): Wait times and Sync delays vs. device transmission range.** (*1 time unit in simulation $\simeq$ 1 ms of real time*). Upper end shows max.

Figure 3.22 shows Sync delay and Wait time. Sync delay is the handover time from one finishing routine to the next waiting routine (same as Section 3.7.1). *Wait time* is the time from when a routine is triggered until it executes its first command—this metric measures CoMesh’s overheads that may delay the start of a routine after it is triggered. With a

sparse topology (40 ft transmission range), wait time is 0.22 s at median (2.55 s max), while sync delay (handover time from routine to routine) is 1.76 s at median (8.89 s max). For dense topology, wait time median is 0.14 s, and sync time median is 1.5 s. Higher transmission ranges of 100 ft and 120 ft only slightly decrease these metrics, e.g., for sparse topology, wait time medians are 0.2 s and 0.2 s respectively, and sync delay medians are 1.7 s and 1.62 s respectively. We conclude that CoMesh scales well in a real building under a real routine workload.

3.8 Related Work

IoT Application Management. IoT application management systems mostly use a central control plane, e.g., HomeOS [111]. Rivulet [67] is distributed and uses smart devices to exchange messages and backup state to shadow nodes, focusing on fault-tolerance. IoT fault-tolerance works also include WuKong [112], but similar to Rivulet, it does not manage routine execution. To the best of our knowledge, CoMesh is the first decentralized IoT routine management system. Finally, while smart buildings are not delay-tolerant networks (DTNs) [113], CoMesh’s design addresses many concerns that also arise in DTNs, namely churn, failures, etc.

Consistent Hashing & Committees. Peer-to-peer (P2P) systems such as Chord [98], Dynamo [99], employ consistent hashing. to find the node storing a key-value pair. CoMesh’s coterie mechanism is inspired by consistent subgroups used by the election protocol in [114], but that mechanism is neither dynamic, nor intended for IoT settings, nor has locality.

Quorums and Agreement. Quorums have been used for coordinating sensor networks’ sleep schedules [115], localization [116], etc. Agreement protocols for wireless topologies include Wireless Paxos [117] and wireless sensor network Byzantine consensus [118]. These require many message exchange rounds, but CoMesh’s coterie requires only one round and ensure safety. Blockchain- and edge-centric consensus schemes [119], [120], [121] have been

proposed for access control, authentication, but not in the context of routine management.

Load balancing. Load balancing has long been studied in internet-based, P2P, and cloud distributed systems [122], and for the edge [123]. In ad-hoc/mesh networks, load balancing has been explored for power-aware routing [124]. Load balancing has also targeted edge-cloud [125] or edge-fog hybrids, like EdgeMesh [126]. Locality-aware consistent hashing has been used in serverless settings [127], unlike CoMesh.

3.9 Summary

CoMesh demonstrates that a fully-decentralized Sense-Trigger-Actuation control plane for large commercial edge IoT meshes is both theoretically sound and practically deployable. Raspberry Pi deployments show that CoMesh reduces per-device CPU load by over $6\times$ compared to a three-hub baseline, cuts inter-device bandwidth by $10\times$ versus a central hub, and achieves median coterie operations (quorum reply, leader election, state transfer) in under 2.1 seconds. Large-scale simulations confirm that CoMesh maintains stable latencies under up to 20% device churn and 5% packet loss, and scales to hundreds of devices without degradation. Today’s emerging IoT standards—Matter [9], Thread [128], and others—align with CoMesh’s vision of a truly autonomous, cloudless control plane for mesh-based edge environments. Future work can explore Byzantine faults, extreme mobility (e.g., vehicular networks), and city-scale deployments. Highly mobile deployments—e.g., large factories where humanoid robots move between stations and roam the floor—are a particularly interesting target: CoMesh can already absorb such movement *reactively* through its churn- and mobility-handling coterie migration, but predicting robot motion (e.g., from task schedules or learned trajectories) would let CoMesh *proactively* reshape coterie before a robot moves, keeping coordination local to wherever the robot is about to be.

Chapter 4 Control in Context: How Smart Home Users Navigate Proxy-Based Control Schemes

This chapter focuses on residential smart homes in the smart space taxonomy of Table 1.1, addressing the *human interaction plane* of smart-space control introduced in Section 1.1: *how* do occupants view and use control interfaces? We present **Control in Context** [16], a mixed-methods study of how and why smart home users navigate the spectrum from physical device interaction to proxy-based centralized control across everyday tasks, failures, and evolving home setups. We surveyed $N = 60$ smart home device owners and conducted follow-up semi-structured interviews with $N = 8$ participants, finding that control is a fluid, reflective process shaped by tradeoffs between convenience, reliability, and cognitive load. Smart home platforms today are designed around vendor ecosystems and commercial integration goals, not around the actual, context-dependent ways people live with and rely on their devices—leaving users to build their own workarounds, absorb switching costs, and troubleshoot failures that the system makes no effort to surface.

This chapter is organized as follows. Section 4.1 introduces our research questions and motivates the study. Section 4.2.1 surveys related work on smart home control and user studies. Section 4.3 describes our survey and interview methodology. Section 4.4 presents our findings, covering control scheme usage, user tradeoffs, and troubleshooting workflows. Section 4.5 discusses design implications for future smart home systems. Section 4.6 covers limitations and future work. Section 4.7 concludes.

4.1 Introduction

The adoption of over 21 billion Internet of Things (IoT) devices has reshaped homes, workplaces, and industries [1], [5], [129], [130], [131]. Vendors of smart devices—e.g., Philips Hue, Ecobee, Apple HomeKit, Siemens, Samsung SmartThings, GE Cync, Honeywell, and others [132], [133], [134], [135], [136], [137], [138]—have traditionally provided their own software for users to control their devices (through *vendor-specific* interfaces), building on

top of existing physical controls like buttons or switches [139], [140]. Yet, as smart homes scale, controlling each vendor’s devices separately becomes less desirable for homeowners.

To combat the overhead of controlling each device or product individually [141], some vendors introduced new modalities of control, layering additional control options on top of traditional vendor-specific interfaces. This allows users to control both vendor-specific and cross-vendor devices through a single interface, potentially reducing overhead. For example, hubs or apps like Google Home [30], Apple HomeKit [142], or Amazon Alexa [29], enable users to control many devices (e.g., lights, thermostats, door locks) through both individual commands and automations (i.e., routines) from a single dashboard [143], [144], [145]. Using such a hub, a smart home user can leverage proxy-based control to create a “good-night” routine that turns off their porch lights and locks their door with a voice command, even when these devices come from different vendors, without operating each device individually [146].

Previous research has established that vendor-specific and cross-vendor control are on opposite sides of a smart device control spectrum [147], [148]. However, these control schemes may refer to *how* users issue commands [149], [150], *which* apps or ecosystems they use [151], or *where* the configuration logic resides (per-device rules or shared routines) [152]. In this chapter, we focus on the *locus of execution* of control logic, namely whether device configuration is initiated directly on the device, or through proxies. Rather than treating one side of the spectrum as vendor-specific and the other as cross-vendor, we observe that these points are parts of an extended spectrum from **physical control**, i.e., interacting with the target device’s physical controls, to *varying levels* of **proxy-based control**, where user commands are routed through one or more intermediaries, such as hub apps, voice assistants, cloud services, etc., before reaching the target device [148], [153]. For example, centralized control is one form of proxy-based control: a single proxy (or a tightly coupled set of proxies) orchestrates many devices and brands from a unified configuration layer [154].

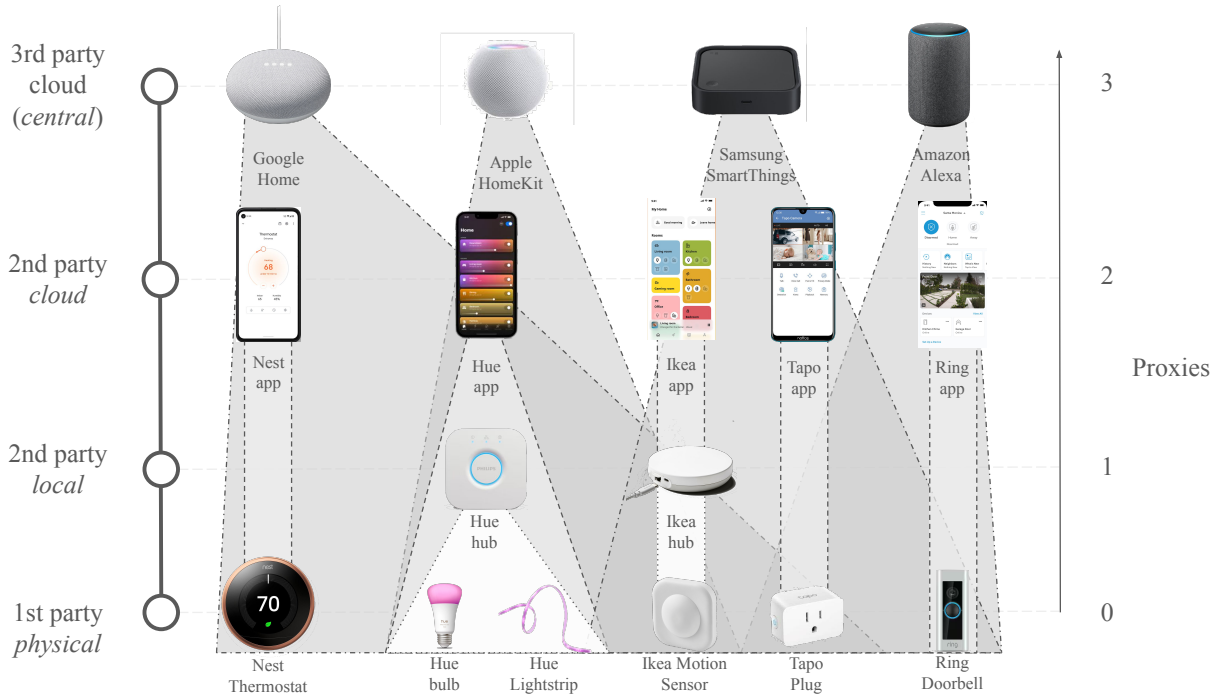


Figure 4.1: **Proxy-based Control Spectrum.** *Control schemes vary along two related axes. (1) Locus of execution: controls may be first-party/physical (direct, on-device), second-party/local (vendor hubs or LAN apps), second-party/cloud (vendor-operated cloud services), or third-party/central cloud ecosystems (cross-vendor platforms such as Google Home or Apple HomeKit). (2) Proxy depth: shown vertically, this indicates how many intermediary layers—hubs, vendor clouds, or central platforms—sit between the user and the device in the control path. Each control scheme has a different reach; the higher the number of proxies, the higher the degree of control centralization (breadth).*

This control spectrum (Figure 4.1), with physical control to proxy-based⁶ central control as the two ends, is widely used in practice, as many homeowners rely on intermediate points on this spectrum rather than its endpoints. Smart home device vendors offer control interfaces that can access either a single device, multiple devices from the same brand, or devices across various brands. People routinely blend proxy-based and physical device control, moving across the locus of execution spectrum based on convenience, reliability, and their understanding of how the system works. These movements become especially visible during troubleshooting. When routines misfire, devices fail, or connectivity is unstable, users abandon usual cross-vendor proxies, fall back

⁶Our use of this term in this chapter is inspired by literature in the systems and networking space, which establishes the use of the “OSI Model” as an abstraction that encapsulates seven layers ranging from physical to application [155].

to device-local physical interfaces, or switch among apps and platforms [156], [157].

Troubleshooting exposes users’ understandings of the smart home infrastructure, including how they choose to reveal or hide connections between devices, apps, and services. These user actions are often referred as *seam practices* [158]. Triage of device errors is a core part of living with smart homes and strongly shapes trust, preferences, and long-term configuration strategies [159].

Despite the known existence of this control spectrum, much of the research on building operating systems (OSs) for smart homes still *assumes* proxy-based control is the primary way for users to interact with and control their home [66], [90], [111]. This assumption of “always proxy-based control” echos how smart home OSs have traditionally been designed for machines [160], [161], [162], [163], [164], [165], [166], datacenters [167], [168], [169], and networks [170], [171], domains where end-users rarely interact directly with individual devices. Empirical studies supported this disconnect by highlighting users experiencing difficulties when managing multiple dashboards for the same task [141]. In occupied homes, however, different loci of execution offer distinct capabilities and trade-offs for users, making the “always proxy-based control” approach incomplete. In addition, much prior smart home control research focused on specific devices or interaction styles instead of empirical user studies [172], [173], [174], [175].

User studies are critical for designing usable and technologically sound systems, complementing prior work that focused mainly on system proposals [65], [90], [176], [177]. Moreover, existing research often abstracts away the messiness of real-world use, where devices are added gradually, apps and platforms overlap, cloud services change, and users inherit configurations set up by others. As a result, we know far less about how existing systems actually behaved in occupied homes and how people patch around misalignments across different control schemes.

This chapter seeks to rectify this mismatch by understanding: *How do users make decisions of which control mode(s) to use across different scenarios?* We investigate

real-world workflows for interacting with smart home devices with and without proxies, giving particular attention to troubleshooting and recovery. We conducted a survey and follow-up interviews to understand how people combine direct device control and proxy-based control (including centralized hubs), and how breakdowns and uncertainties shape these choices. First, we examine how people navigate and combine different control schemes in everyday use: when and why they prefer proxy-based hubs, per-brand apps, or device-local control in different contexts. Second, we focus on troubleshooting as a critical moment within that broader experience. During troubleshooting, users’ understandings of the system surface more clearly and they often reconsider which control loci they trust and are willing to invest in.

By investigating real-world workflows for interacting with smart home devices, we answer the following research questions:

- **RQ1:** *How do smart home users navigate and combine different control schemes in their homes?*
- **RQ1a:** *What factors drive users to move between, and combine, control methods within their homes?*

We also examine how device and interface breakdowns or errors affect the above choices:

- **RQ2:** *When smart home interactions do not go as expected, how do user perceptions of control schemes shape their responses?*

Sub-questions arise about how these experiences affect perceptions:

- **RQ2a:** *How do troubleshooting experiences affect users’ perceptions and trust of/in their preferred smart home control methods?*
- **RQ2b:** *How does user uncertainty around functionality affect their strategies for engaging with different control methods and their seam practices?*

In summary, this chapter makes the following contributions:

- We evaluate how smart home control affects a user’s ability to meet their smart home goals, and find that *the spectrum of smart device control plays a major role* in user preferences and experiences in managing and using their smart home devices.
- We reveal how and why users decide to control their devices at different points along the proxy-based to direct device control spectrum, exposing the different seams within their smart homes, understanding that *users move freely along the spectrum, guided by situational factors such as convenience, task, and perceived reliability*.
- We characterize troubleshooting workflows and seam practices across control schemes, and detail how breakdowns and uncertainties prompt users to *reconfigure control arrangements, fall back to direct device control, and strategically reveal or hide infrastructural seams when troubleshooting*.
- We identify and understand the factors that guide smart home control preferences and suggest ways for future research and design to incorporate these perspectives, namely *accounting for spatial and temporal heterogeneity* in smart homes, as well as *supporting movement through varying seam hiding/revealing practices along the smart home control spectrum*.

4.2 Related Work

We build upon prior literature on user experiences within smart homes and associated control modalities.

4.2.1 Smart Homes, Devices, and Control

We define *smart home devices* as “*sensors, monitors, interfaces, appliances, and devices networked together to enable automation as well as localized and remote control*” [178].

These include but are not limited to smart lights, smart home security systems, smart thermostats, smart TVs, and smart speakers [179]. We use the term *smart home ecosystem* to describe the interconnected set of devices, platforms, applications, cloud services, data flows, and household members that together provide and control smart functionality in a home. An ecosystem can span multiple brands and control loci (e.g., hubs, vendor apps, device-local interfaces), and changes as devices are added, removed, or reconfigured over time.

As smart devices become integrated into homes, user interaction and control over these systems emerge as key areas of study. Integrating smart devices into households is a domestication process involving pilot users to drive the setup and passenger users adopting these devices, sometimes even without understanding them [180]. Household members gradually adapt their routines around these devices, exercising control over how smart devices fit into their daily lives [181]. Research highlights that users can have different expectations of smart device behaviors when two routines run simultaneously because they have different values [182]. These devices, marketed for convenience, inherently come with user expectations regarding ease of use and seamless integration [183], [184]. Additionally, studies revealed that stakeholders like cohabitants, bystanders, and non-users could not control how they interact with smart home devices placed in different locations in households [185]. Users tend to group their devices based on their location, functionality, and/or use contexts to expedite control [175]. A common thread among this prior research is the central role of control in shaping individual experiences with smart home devices [186]. Despite the growing variety of smart home devices and control methods, there is limited literature examining *how* users navigate control schemes for individual devices or groups of devices within their homes. In this chapter, we identify key factors that shape device control workflows and explore how users' experiences, including troubleshooting, influence their perceptions of these control schemes.

4.2.2 Smart Home Control Schemes

Table 4.1: **Prior work mostly assumes one control scheme of the spectrum exclusively. It does not study the relationship between the two, even when both are discussed in the same work.**

Papers	Focus	Contribution
[149], [175]	Central, limited Physical	Empirical
[111], [187]	Central	Empirical
[65], [90], [188], [189], [190], [191], [192], [193], [194], [195]	Central	System
[196], [197]	Central	Literature Review
[198]	Cloud, Local, Physical	Evaluation
[199]	Central	Evaluation
[176], [177]	Second party	System
[200]	–	Empirical

Smart home control schemes refer to the guiding paradigm for categorizing mechanisms to control smart home devices. In this chapter, we focus on the spectrum of *proxy-based control*, visualized in Figure 4.1. The spectrum represents the levels of centralization or indirection in smart device control. Essentially, *proxy-based control* refers to how user commands are sent to devices through a proxy, such as an application, voice assistant, routine/automation, etc. On one end of the spectrum sits *physical device control*, which refers to the direct interaction between the user and target device (first party), without any intermediaries or indirection (i.e., zero proxies), regardless of the interface modality, i.e., touch, voice, etc. On the other end of the spectrum, *central control* (or *third-party cloud control*), represents the highest level of centralization, with multiple layers or proxies of control systems between the user and the target device. *Central control* is an extreme type of proxy-based control, where a single interface allows users to control multiple smart home devices, possibly of different brands, directly or indirectly. The points in between the two

extremes refer to vendor-specific (second-party) interfaces, or per-brand control, that execute locally in the same network as the target device through Zigbee, Bluetooth, etc., or in the cloud, adding one or two network hops to the control path, respectively.

Our focus on this spectrum of control builds upon existing theoretical HCI models. In this chapter, we focus on the information ecology model in smart homes. Information ecology theory envisions smart homes as continuously evolving technologies that must interact with existing human and technical processes [201], [202], [203]. Our control spectrum refines this view by specifying *where* actions are executed in the ecology (the locus of execution) and *how* seams in infrastructure are hidden or revealed during everyday use and troubleshooting. The control spectrum also reinforces concepts introduced in other models used often in user-centered smart home research, such as the Technology Acceptance Model (TAM) [204], [205], [206], [207] and Activity Theory (AT) [208], [209], [210].

Smart devices can be controlled through varying levels of these two types of control, which users must understand and navigate to build their desired smart home setup [175]. These reflect common methods of controlling smart devices, including smartphone applications, voice assistants, smart hubs, routines/automation, and physical controllers in the market today. We detail part of the landscape of existing research surrounding smart home control in Table 4.1.

Such an abstraction reduces complexity for smart home device users and provides a single, cohesive user experience. Apple Home⁷, Google Home⁸, and Samsung SmartThings⁹ are examples of central control schemes. Across systems, security and HCI, research focuses on centralized control, such as surfacing the need for finer-grained access control within households [146], designing more transparent and privacy-focused data processing for centralized smart home control [211] or proposing a unified interface where applications,

⁷<https://www.apple.com/home-app/>

⁸<https://home.google.com/welcome/>

⁹<https://www.samsung.com/us/smartthings/>

commonly known as routines, interact with different smart home devices [111]. In contrast, physical control allows users to maintain coarse-grained control over their device even in the face of network failures [149]. A downside is that users must maintain multiple applications in their smartphone if their smart devices come from different brands, which quickly becomes exhausting [175].

Early research on smart home devices shows that direct and unmediated interactions with devices are often perceived more positively than interactions mediated by a single source of indirect response [212]. This suggests that users’ preferences may vary depending on whether they interact with devices through a highly centralized *proxy-based* interface or use control schemes with fewer proxies involved, yielding more direct feedback. Matter [213] has been introduced in an effort to bring central control to the local network and reduce the intermediaries, yet vendors still expose only certain APIs through that integration, making little to no difference from the interface perspective. Few studies have examined the dynamics between these control schemes. As shown in Table 4.1, prior work primarily focuses on a particular end of the spectrum. Prior work has examined reasons why users may interconnect or purposefully separate their smart devices [175], [214]. However, such research focused on specific tasks in a controlled environment, limiting its ability to capture the challenges users face in real-world scenarios, for example, when devices fail. These troubleshooting experiences play a crucial role in shaping users’ perceptions of control schemes along the spectrum, which we explore in detail in the next section.

4.2.3 Troubleshooting Smart Home Devices

Troubleshooting, the process where smart device users try to identify and resolve errors within smart devices, is a common part of the smart home device management experience [159]. It contains several different steps and involves working with both software, i.e., device applications, and hardware, i.e., the physical device [215]. Device failures erode expectations and trust among smart device owners, as they have high expectations of how

the device will improve their lives at home when they initially purchase it [159], [216].

A seamless troubleshooting experience, where users can easily isolate and address issues, can increase positive perceptions of the device [217]. However, it is often during the troubleshooting process that the boundaries and imperfections, i.e., the seams, of smart home devices are exposed to the user. Research has identified seams as both something that must be hidden from the user [218], [219], as well as an opportunity to support enhanced user understanding of their devices [220], [221], [222]. We do not assume that either perspective on seams is correct; instead, we investigate how smart home device control schemes affect the seam hiding and revealing practices of devices and users.

Schemes along the control spectrum can offer drastically different troubleshooting experiences and seam-hiding practices, affecting user perceptions of these control schemes differently [223]. While it may offer increased convenience, central control can be especially challenging due to only a single point of failure [224]. When an error or exception occurs in the central controller, all devices controlled via central control are at risk of failing. In addition, when a single device connected to the central controller fails, the user must look not only at the failing device but also at the central controller to find the source of the issue. Direct device control provides a more localized approach to troubleshooting, allowing users to examine only one device at a time. However, in large-scale home failures, such as power outages or network failures, such control schemes require the user to troubleshoot each device individually. Control schemes involving very few proxies, such as second-party proxy-based control, or *per-brand control*, often offer more fine-grained functionality than other schemes, making troubleshooting more efficient and intuitive. When a Roomba stops mid-clean, the iRobot Home app shows the exact error code¹⁰; a central controller like Alexa/Google typically only starts/stops the vacuum or reports ‘device not responding’¹¹.

While previous work has focused solely on the technical aspects of troubleshooting [225], [226], this chapter investigates how users’ choice of control schemes

¹⁰<https://homesupport.irobot.com/s/article/21044>

¹¹<https://www.amazon.com/gp/help/customer/display.html?nodeId=GJRYAWXYCQKQF28E>

Table 4.2: **Summary of Survey Participant Demographics.**

Age	18-24 (2%), 25-34 (28%), 35-44 (23%), 45-54 (31%), 55-64 (7%), 65+ (9%)
Education	High School or GED (16%), Some College (22%), Associate’s Degree (7%), Bachelor’s Degree (42%), Post-Graduate Degree (11%), Some High School (2%)
Gender	Male (40%), Female (60%)
Race	Black/African American (4%), White (92%), Asian (2%), Hispanic or Latino (2%)
Living Situation	House (88%), Apartment (10%), Town Home (2%)
Number of Smart Devices Owned	0-5 (16%), 6-10 (19%), 11-15 (31%), 16-20 (21%), 21-25 (7%), 26-30 (2%), 31-35 (2%), 36-40 (2%)

in smart homes affects their troubleshooting experiences. We extract generalized troubleshooting workflows grounded in real-world user experiences. Additionally, we discuss with users how these troubleshooting experiences have affected their perceptions of their devices and the ways they control them. This perspective allows us to connect concrete troubleshooting workflows to broader theories of seams and infrastructure, and to derive design implications for how future systems might better support these transitions.

4.3 Methods

We conducted a survey and follow-up interview study to understand the perceptions of smart home device users and their control schemes. We collected $N = 60$ survey responses through Prolific, a commonly used crowdsourcing platform for collecting data for online human-subjects research [227], [228], [229] (Table 4.2). We conducted follow-up interviews with a subset of $N = 8$ survey respondents to extract specific insights between September 2023 and March 2024. Our study’s methodology is grounded in previous user-centered smart home research, which focuses more on eliciting high-level themes rather than drawing statistically significant conclusions [230], [231], [232]. All study procedures were piloted internally and approved by our institution’s Institutional Review Board (IRB).

4.3.1 Survey Methodology

Participants first completed a brief screening survey (median completion time: 65 seconds), which required them to provide their Prolific ID and indicate the number of smart home devices they owned.

Our goal was to study people with active smart home setups rather than casual single-device owners. We therefore restricted survey participation to adults who (1) were registered on Prolific and reported owning at least one internet-enabled device in Prolific’s prescreening questions [233], and (2) reported owning at least four smart home devices (e.g., smart lights, plugs, thermostats, cameras, speakers) in our screening survey. Participants were not limited to central-hub households, and those only using per-brand apps or direct device control were eligible. 60 participants completed this survey.

Participants were paid \$0.40 USD (hourly rate of \$24 USD) for completing the screening survey and \$4.00 USD (hourly rate of \$16 USD) for completing the main survey. We did not specifically recruit or target professional smart home installers or technicians. A small number of participants described installing devices themselves (e.g., DIY setups), but they still participated in the study as end users of those systems rather than in a professional capacity.

We had 43 valid survey responses from the 60 participants, excluding those who failed any of the arithmetic-based attention-check questions we included at random points in our survey (e.g., “What is $2 + 7$ ”) [234] or had inconsistent responses, such as differences in the number of or types of smart devices owned. All 60 survey participants were compensated for their time.

4.3.2 Survey details

The survey aimed to elicit participant preferences and perceptions related to smart home device control, as well as existing troubleshooting workflows and how device control preferences influence them. The full survey is included in Appendix A.1.

General Smart Home Setup. The survey began by collecting information regarding each participant’s general smart home setup. Participants listed smart home devices they currently use, where they lived, where each device was located, and what smart home device-related applications they currently have on their phones.

Participants reported what, when, and why they purchased their most recent smart home devices, then rated how well these devices met their goals and explained their ratings. We asked participants to focus on this single device to reduce survey cognitive load [235] and provide detailed context.

Current Methods for Device Control. The survey then asked participants how they *controlled* the various smart devices in their home. Participants selected their control schemes for a specific device in their home from the following: routines, central control, per-device control, voice commands, and content streaming/casting. Participants were asked more broadly about their smart home control patterns, such as how they choose to control a device in a specific way when it could be controlled in multiple ways. This section concluded with participants reporting their awareness of home management systems, such as Amazon Alexa, and how they were made aware of these systems’ existence.

Device Failures and Troubleshooting Methods. To understand device troubleshooting experiences (RQ2), the survey asked participants to describe issues they encountered with their smart devices. Participants recalled when their smart devices did not work as planned and their perceptions of the devices during the failure. They also indicated whether they had ever returned a smart device because it stopped working as intended. Participants then provided a step-by-step account of how they troubleshooted the device’s failure. They continued to report whether they had ever given up on getting a smart home device to work in their home, identifying the device if so.

Survey Wrap-Up and Interview Interest. Finally, the survey asked whether the participant was the primary user of any homes other than their primary residence. If they were, participants were asked if there were any significant differences in how they

Table 4.3: **Smart device ownership of our 8 interview participants.** For each smart device, an ‘x’ indicates that the participant owns that device in their home.

	Smart Plug(s)	Smart Light(s)	Smart Thermostat(s)	Smart Door/Window Sensor(s)	Motion Sensor(s)	Smart Smoke Alarm(s)	Smart Camera(s)	Smart Security System(s)	Smart TV/Casting Device(s)	Smart Speaker(s)	Smart Tag(s)	Smart Health Device(s)	Smart Washing Machine/Dryer(s)	Smart Dishwasher(s)
P1	x	x	x				x	x	x	x		x		
P2	x	x	x			x			x	x		x		
P3		x	x	x		x	x	x	x				x	
P4			x	x	x		x	x	x		x			x
P5	x	x	x		x		x		x			x	x	
P6	x	x					x		x					
P7	x	x	x		x					x		x		
P8			x				x			x		x		

controlled these devices compared to their primary residence. Then, participants were invited to provide their email if they were interested in participating in a follow-up interview with research team members.

4.3.3 Follow-up Interviews

40 survey participants expressed interest in the semi-structured follow-up interview in exchange for an extra \$20 USD in compensation. We prioritized invitation to respondents with the largest reported smart home devices they owned, with the aim of capturing rich experiences across the control spectrum in multi-device ecosystems. Although later waves extended invitations to all participants eventually, only a subset responded, and we conducted 8 interviews. Interview participants had a variety of devices in their homes, allowing us to elicit a range of perspectives (Table 4.3). We used interviewees’ survey responses to personalize specific questions during the study. For example, if a participant reported owning smart lights, we asked how they controlled them and why. Each of the 8 interviews we conducted lasted around 30 minutes and was structured into the following sections (the full interview protocol is included in Appendix A.2):

- *Smart Home Setup*: Participants were asked to describe their most recent smart device purchase, their motivation for purchasing it, how they set it up, and whether it met their intended goal.
- *Smart Home Device Control*: Participants discussed how they controlled their smart home devices. Devices they mentioned in the survey were brought up to

contextualize the discussion (Section 4.3.2). Participants then described their ideal control scheme for their smart home devices. Participants could mention any control method and scheme that may or may not exist on the market. Participants described whether they controlled smart home devices in a way that allowed them to work together, i.e., through routines. Participants discussed how this setup was beneficial to them, if they did, and how they managed it, i.e., what method they used for central control. Finally, participants reflected on how their existing setup and preferences for smart home control methods affected their willingness to purchase new types of smart home devices.

- *Smart Home Errors and Exceptions:* Participants described specific experiences when their smart home devices did not work as planned. This section asked participants to elaborate on experiences they had briefly described in the survey and to describe any errors or exceptions not included in the survey. In particular, we asked participants to explain any errors they had when integrating devices into their smart home infrastructure.
- *Smart Home Troubleshooting:* The interview concluded with participants describing their troubleshooting workflow in detail and providing examples of when it was successful and when it was not. The detailed troubleshooting workflows and the impact of troubleshooting experiences on their perception of smart home devices and the corresponding control scheme were augmented by the participants' survey responses.

4.3.4 Data Analysis

We conducted descriptive statistical analysis of the survey data, calculating mean and median ratings for Likert scale-based questions. For qualitative responses and interviews, we performed in-depth qualitative analysis using thematic analysis and open coding [236],

[237]. Two members of the research team independently coded four interviews, followed by cross-coding two interviews to assess inter-rater reliability (IRR) [238], resulting in a Fleiss' Kappa of 0.44, indicating moderate agreement [239]. Codes were consolidated and discussed through team discussions, yielding codes and themes directly related to the research questions, as well as a concrete set of user workflows for troubleshooting smart home devices. These results provided high-level insights into user perceptions, user behavior, and control workflows for smart home devices (Table 4.4).

While our interview and survey samples are not designed to support statistical inference, our findings should be interpreted as analytic generalizations about how smart home users navigate proxy-based and physical device control, not as precise estimates of how common each behavior is in the broader population. Our work sits among other qualitative smart home behavior research, providing in-depth insights into user experiences, privacy concerns, and adoption barriers [240], [241].

Table 4.4: **Codebook generated from qualitative analysis.**

Theme	Codes	Meaning
Control Scheme Perspective	Central control: viewed positively or negatively Central control: viewed as useless or convenient Per-device control: viewed positively or negatively Per-device control: viewed as useless or convenient	Participant describes how they view the described control scheme
Awareness Mechanism	Advertisement Online research Social media Word of mouth	How participant became aware of smart home device control scheme
Usage Pattern	Safety Convenience Cleaning Communication Entertainment	The described reasons for why participant uses a smart home device
Usage Frequency	Daily Weekly As needed	How often the participant uses the device
User Type	Primary user Partner/Spouse Child Pet Secondary/Passenger user	The type of user in the participant's home that benefits from the described smart home device
Perception-driven Behavior	Increased device usage due to positive perception Decreased device usage due to negative perception Avoiding device type/brand for future purchases Seeking out device type/brand for future purchases Considering integration for future purchases	How the participant's perception of the device affects their behavior
Device Control Factors	Convenience Increased functionality Trust Device types	Participant reasons for using a particular device control scheme
Troubleshooting Steps	Check and reconnect physical device Check home management system Check application Contact IT support or conduct online research Check and restart home Wi-Fi Modify device commands or routines Give up	How participants troubleshoot their devices
Troubleshooting -based Perception	Positive experience improving device perception Negative experience worsening device perception Troubleshooting experience reduces trust Troubleshooting experience improves trust	Effects of troubleshooting experience

4.4 Results

Our analysis of survey and interview data focuses on how users navigate the spectrum between proxy-based and physical device control in everyday smart home use. We structure our findings around three core themes: (1) user strategies for navigating this spectrum (RQ1), (2) the tradeoffs they perceive when doing so (RQ1a), (3) how navigation strategies, perspectives, and trust shift when users encounter device errors or failures (RQ2, RQ2a), and (4) how users commonly struggle to realize their goals across the control spectrum (RQ2b).

Sample and device ownership. Our survey respondents were experienced smart home users: they owned a median of 12 smart devices (Table 4.2). The most common reasons they gave for their most recent purchase were upgrading a device already in the home (38%), added convenience (19%), added control over some home process (19%), and added security (17%), and 90% used that device at least three times a week. Participants were largely satisfied with these purchases: 67% strongly agreed that they accomplished the goals they had in mind when buying the device, most often attributing this to added control (42%), convenience (37%), and security (37%). This grounds the navigation behaviors below in the experience of users who actively live with sizable, multi-vendor smart homes.

4.4.1 Control spectrum navigation strategies (RQ1)

Survey and interview responses showed that participants rarely commit to a single point on the control spectrum. About half of the survey participants (51%) controlled their devices using voice commands, which can be used across the proxy-based control spectrum. Additionally, 33% of participants reported using routines to control their devices, which allowed them to control groups of devices simultaneously. Furthermore, 18% of participants reported using ‘content streaming or casting’ to control single devices, which is

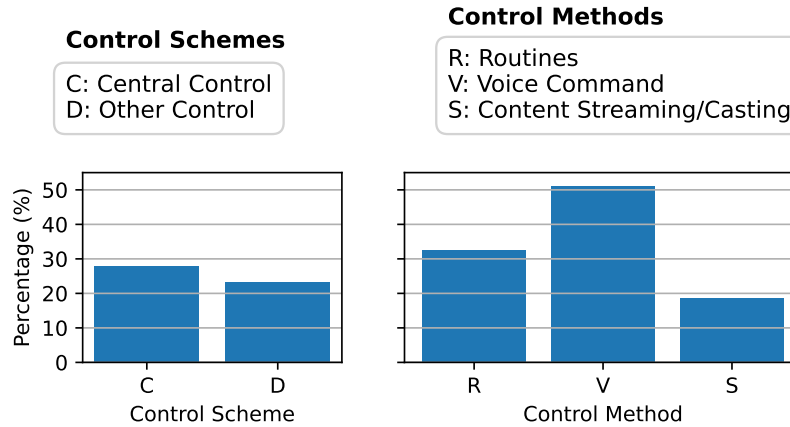


Figure 4.2: **Percentage of participants using each control scheme/method.**

common for media devices such as smart TVs. Both user preferences and device constraints influenced participants' choice of control methods; for instance, participants using Amazon Alexa often relied on voice control due to physical device functionality limitations, while others with broader device ecosystems could select among multiple control methods.

As smart home ecosystems became more complex, participants increasingly shifted toward centralized control by purchasing and using smart home third-party hubs and adopting routines. Figure 4.3 shows that the use of centralized schemes rises alongside the number of devices owned among the survey participants.

Interview participants who manage many devices reinforced this result, as they reported turning to centralized hubs or voice assistants to reduce the cognitive load of juggling multiple interfaces. In particular, interview participant P7, who owned nearly 50 devices, said they relied heavily on central control but acknowledged its limitations: *“It’s not easy to remember [what each device is named].”* Still, this strategy helped them operate a large-scale setup more efficiently.

At the same time, users moved along the spectrum toward physical control when centralized systems failed to meet specific needs. This involved falling back to control methods such as physical interaction with a specific device or using a brand-specific application on their phone. Several interview participants noted that detailed settings,

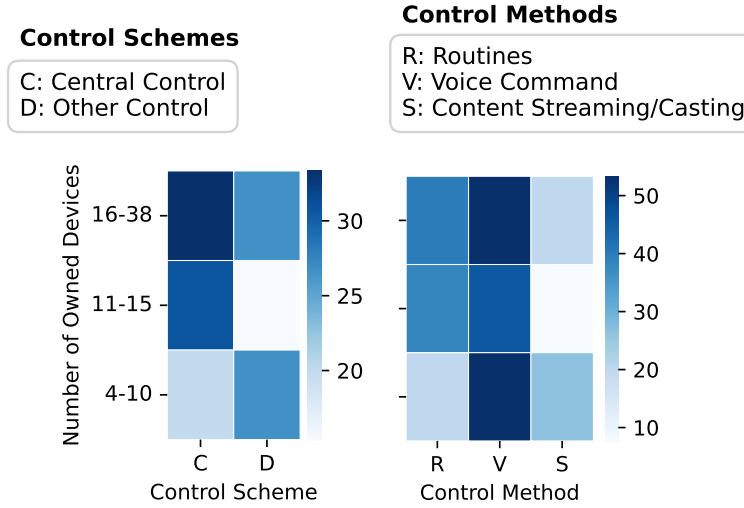


Figure 4.3: **Percentage of survey participants using central or other control schemes and several control methods vs. number of owned devices.** Each cell’s color shows the percentage of participants falling into it who choose the respective control scheme or method. The percentages for the different options of control schemes and methods do not add up to 100% since the participants could provide multiple answers.

such as firmware updates or custom automations, were often only accessible through the device’s native app. For example, P3 shared: “*There is a button in the [Hue] app to make the light bulb that you’re trying to interact with, blink on and off, so that you know exactly which one it is; that doesn’t exist in the Google app.*”

Further, Figure 4.4 shows that non-central control remains popular for devices requiring more granular or custom interactions, such as security systems or vacuums, where direct control over specific features is necessary. We also observe that participants usually chose to control a device exclusively with either routines or voice commands, but not both.

Some participants used strategies to stay where they were on the control spectrum rather than move along it. Half of the interview participants (4/8), in particular, expressed that they were hesitant to buy new smart home devices that required significant effort to integrate into the existing infrastructure. For example, participants often decided against purchasing devices that required them to download a new application on their phone to set up or use the device. They instead chose to purchase devices that could be controlled using

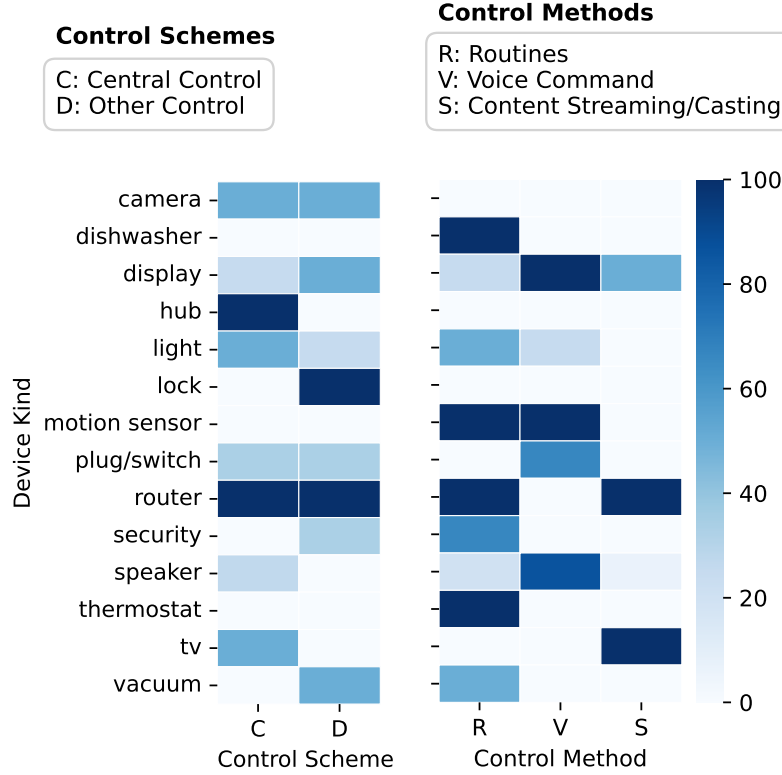


Figure 4.4: **Percentage of participants using each control scheme/method vs. device kind.** Participants may refer to a device with a different description from how they most commonly use it, leading to an occasional discrepancy between the device kind and the control method used; for example, a participant may refer to Alexa as only a smart speaker when it is also a hub [242], [243], [244].

applications already on their phone. P1, when discussing why they purchased a particular brand of new smart device, echoed this sentiment, saying, “*I already got a lot of apps on my phone. So when possible, I don’t wanna put more apps on. So I checked that it’ll work with Smart Life. I checked that it’s compatible with Alexa.*” Similarly, P3 described the cost of integrating a new device into their existing smart home control scheme as a significant factor, saying “*I definitely consider that before buying a potential product, and if possible, I try to buy ones that don’t require me to use a new app or ecosystem. There is a cost of switching there that I don’t necessarily want to introduce to myself.*” Participants’ responses reflected a desire to gain the benefits (e.g., safety, convenience, etc.) of new smart devices in their home without the potential drawbacks of needing to work outside their home’s

existing smart device ecosystem, forcing them to adopt mindsets of “brand loyalty” [245].

This app-minimizing behavior is also visible at the level of the whole survey sample. Figure 4.5 relates the number of smart home apps a participant has installed to the number of devices they own. While users tend to accumulate more device-specific apps as their ecosystems grow, most kept only 1–4 apps—roughly 73% of participants had at most four—suggesting an effort to minimize cognitive load and to centralize control when working with multiple vendors rather than installing yet another device-specific app.

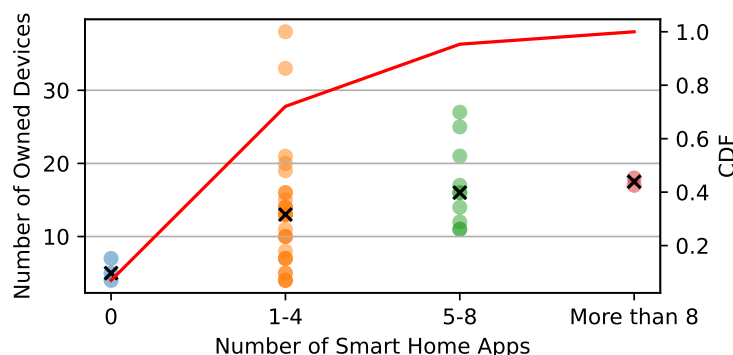


Figure 4.5: **Survey: number of smart home apps installed vs. number of devices owned.** Points are participants (black X = per-group median); the red line is the CDF over the number of installed apps. Most participants keep only 1–4 apps even as device counts grow.

Long-term use affected user strategies as well. P3 found that as time passed, the scale of the smart home increased, and it “no longer would be easy to control with just [their] phone,” enhancing the need for more fine-grained control and more options through more direct control schemes. In contrast, P5 is “using two providers, Nest and Tapo. I want to choose only from those two the new device,” and limit the number of interfaces they are using, even if their smart device collection grows. Further, expertise grows over time for some, such as P2: “for setup, I know now what to do. I’ve had problems in the past. And I’ve learned from those problems,” letting them more comfortably approach new interfaces. Yet, some interfaces are still particularly hard to get used to, as P3 describes having to relearn how to find the routines setup, eroding the trust and patience that users initially have for smart features that centralize control: “I don’t rely on routines for most of the

Table 4.5: **Survey: common reasons participants gave for why they controlled a device in a particular way.** $N = 43$ valid survey responses.

Reason	Frequency (%)	Example quote
Ease of use	8 (19%)	<i>“It’s the simplest and easiest way to use.”</i>
Location of device	9 (21%)	<i>“If I am nearby I use touch controls, otherwise voice commands.”</i>
Maximize device control	13 (30%)	<i>“Complete control over the entire system, without using 10 apps.”</i>
As a backup plan	12 (28%)	<i>“If Alexa isn’t working to ask to turn on we can use the app.”</i>
Force of habit	4 (9%)	<i>“Voice commands are easier than to hunt out the app on my phone.”</i>

house, except for the room where our animals are.” These experiences show how temporal changes in smart home scale or expertise push users to expand or preserve their navigational limits on the control spectrum.

Overall, most participants did not adopt a fixed position on the smart home control spectrum. Instead, they moved fluidly along it, adapting to the demands of scale, task complexity, and reliability using various strategies. In the next section, we examine the specific tradeoffs users perceived and how these influenced their decisions to stay at a certain point or move within the control spectrum.

4.4.2 Control spectrum navigation tradeoffs (RQ1a)

While participants demonstrated flexibility in their control strategies, their decisions were shaped by tradeoffs between convenience, precision, cognitive effort, and integration. These tradeoffs were not static; instead, they shifted with the scale of the smart home, the types of devices involved, and the user’s comfort with technology. Participants weighed these factors when choosing whether to lean toward centralized control (where a single proxy controls many devices) or physical individual device control at any point in their day.

Table 4.5 summarizes the reasons survey participants gave for controlling a device the way they did, which we unpack in the rest of this section.

Participants commonly cited convenience as the most compelling reason for adopting centralized control. This was particularly true as the number of devices in a smart home increased. For example, interview participant P8, who had a large-scale smart home setup, preferred centralized control because it streamlined routine tasks and allowed control through a single interface: “*It’s just obviously the convenience.*” Survey responses supported this perspective, showing a clear increase in central control-based methods’ use among participants with more than 15 devices (Figure 4.3). Centralized interfaces reduced the friction of opening and switching between multiple apps and offered the possibility of controlling the whole house with fewer steps. In some cases, these tradeoffs were negotiated within the household rather than by an individual. P6 explained that their partner preferred consolidating under Alexa “to keep everything easy and streamlined,” while they themselves “didn’t really care about the brand,” reflecting how household priorities could drive ecosystem choices.

In contrast, control through second-party interfaces, or per-brand control, was often favored for its granularity, clarity, and ease of use, especially when dealing with specific device configurations or troubleshooting. P2 illustrated this distinction clearly: “*Whenever I need to adjust something specific, I go straight to the device’s own app... Alexa doesn’t have those settings.*” P5 echoed this sentiment, noting that certain features—like firmware updates or custom scenes—were only available in the vendor’s own app: “*Google Home doesn’t support [those].*” This division reveals a tradeoff between breadth and depth: centralized control enabled broad coordination across many devices, but per-brand control offered deeper access to individual brand functionality.

Participants also discussed the usability and user experience differences between central and per-brand control. P3 described centralized interfaces as confusing and overly technical, especially for less tech-savvy users: “*If you weren’t very tech savvy, you would not be able to figure it out.*” Meanwhile, they described the per-brand app experience as more intuitive and better designed: “*The Hue app communicates with Hue lights, and that*

integration is a little bit better.” These comparisons underscore how usability and familiarity influenced participants’ willingness to rely on one scheme over another.

Not all tradeoffs were framed as strictly negative. Some participants viewed the coexistence of control schemes as beneficial, enabling fallback options or additional layers of control. For instance, participants appreciated having per-brand apps and physical control as a backup when centralized control failed, or vice versa. As P5 described, they would look at a device’s specific app first for assistance, rather than the central control system app. Similarly, per-brand apps may also fail and trigger another step down to physical control, as P5 describes: “*Sometimes it shows as offline because the [brand] app is not able to connect, but the physical device is already up and running. Then I have to turn it off and turn on or do something to get it back up again.*” In these situations, control scheme redundancy offered resilience, not just complexity.

Overall, participants’ experiences reveal that smart home control is shaped by ongoing negotiation. Users must continually balance ease of use with technical specificity, minimize app clutter while maximizing access, and integrate new devices without overwhelming themselves. These tradeoffs not only shaped how users controlled devices but also influenced how they structured, grew, and maintained their smart home management systems over time. In the next section, we examine how these navigation strategies and tradeoffs shift when smart home management systems encounter errors and how users respond to breakdowns that disrupt their preferred control methods.

4.4.3 Navigating the control spectrum when failures occur (RQ2, RQ2a)

While participants developed personalized strategies for navigating the smart home control spectrum, their routines were often disrupted by errors or device failures. These breakdowns forced users to shift control strategies, reconfigure devices, or adopt new troubleshooting behaviors—often in ways that tested the limits of their preferred schemes. Our findings show that errors were a critical moment when users’ relationships to the

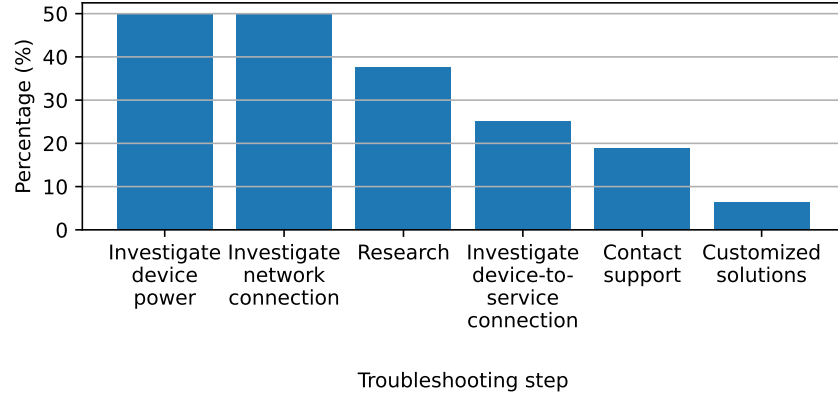
control spectrum changed: participants re-evaluated their trust in control methods, engaged in layered troubleshooting, and sometimes abandoned devices or features altogether.

In the survey, 65% of participants reported that they had experienced at least one issue where a smart device did not work as intended, and 28% reported repeated issues. These issues ranged from connectivity failures (33%), such as devices becoming unresponsive during WiFi outages, to misinterpretation of voice commands (23%) and execution failures, where the device failed to perform the correct task (21%). For instance, one participant noted: “*When there has been an issue with WiFi in the house, the Amazon Echo tends to have issues.*” In these moments, control mechanisms that were once trusted became unreliable.

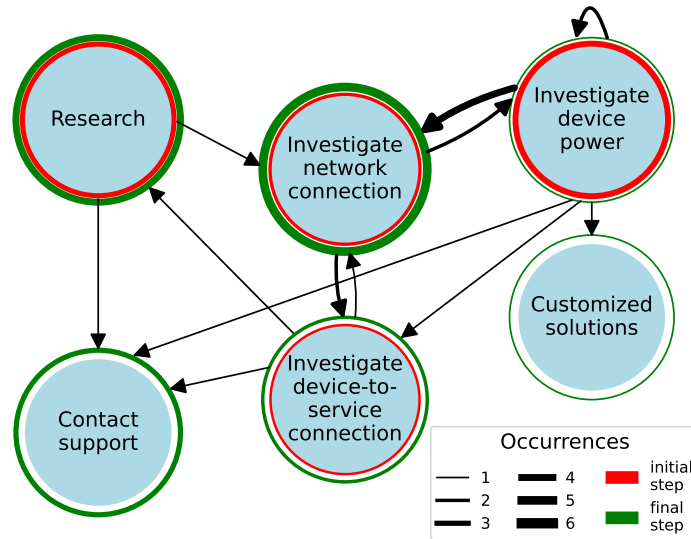
When troubleshooting, participants frequently shifted away from their preferred control methods. For users who primarily relied on central control, their first step was often to check the centralized app or voice assistant interface to confirm whether the command had been sent correctly. Interview participant P7 explained: “*I will go and look at the app straight away and make sure that the heating has activated rather than just trust it.*” However, if the centralized interface provided insufficient feedback or failed to offer detailed diagnostics, participants moved toward physical troubleshooting, such as using the vendor app or physically interacting with the device.

Control schemes with very few to no proxies played a crucial role in this context. Many participants described them as more reliable and informative when devices failed. For example, interview participant P5 said they would open the device-specific app to get a clearer view of what had gone wrong. Others, like P8, relied on physical actions like power cycling: “*And if that doesn’t work, I’ll just switch it off and back on again.*” These tactics were rarely described as first-choice methods but were commonly used once central or automated systems proved insufficient.

Survey data mirrored these patterns. Figure 4.6a shows that 50% of participants restarted the device as part of their troubleshooting process, while 36% restarted or



(a) Percentage of participants utilizing each troubleshooting step. Each participant may only contribute once towards this metric.



(b) Troubleshooting step sequence. A node has a red ring around it if it has been described as an initial step in at least one participant's troubleshooting process. A green ring around a node means it was the final step of one or more participants' troubleshooting process. An arrow links two nodes A, B if at least one participant has gone from step A to step B. The line width of each ring or arrow represents the absolute number of its occurrences across our participants' processes.

Figure 4.6: Troubleshooting steps results from survey responses. *As many are experienced users of their smart home setups, this may subtly shape the emphasis on certain troubleshooting approaches depicted here.*

reconnected their device to the internet. Only 12% reported contacting tech support, indicating that most participants tried to resolve issues independently using a set of well-practiced responses. Figure 4.6b visualizes the common sequences users followed when troubleshooting. Initial steps often included power or connectivity checks, followed by

deeper exploration through brand-specific apps, web research (22%), and more.

Participants also described emotional responses to persistent failures. For some, repeated breakdowns led to troubleshooting fatigue or abandonment. Interview participant P6 shared that a voice-controlled TV consistently misunderstood commands, leading them to eventually “*give up on it for a long time*” before replacing the device entirely. While only 9% of survey participants said they returned a device after an error, and 11% said they gave up entirely on a type of device, these moments marked important shifts in how users related to control schemes and whether they were willing to continue adapting.

Notably, participants’ troubleshooting practices revealed that they often understood control methods as part of the problem space. For instance, P7 mentioned researching whether thermostat failures were due to device issues or miscommunication with their central control platform. When troubleshooting failed, users sometimes restructured their systems—switching platforms, removing routines, or replacing devices that didn’t integrate well with others, as P5 said: “*[My Kasa switch] was not able to integrate correctly. I had to pair it every time with the Google Home App and every time it would disconnect. And then I had to do it again. And it was painful. So I ended up returning the device.*” Similarly, P1 described how after an extended period of troubleshooting, they had “*resigned myself to just using the app.*” In these experiences, participants found themselves forced into a particular control method, rather than having autonomy over how they *wanted* to control the device.

Overall, device failures and errors forced participants to switch between control schemes, uncovering limitations and prompting changes to their routine. While many relied on layered troubleshooting strategies, persistent problems sometimes led to long-term shifts in user behavior, feature abandonment, or restructuring of control ecosystems. These findings show that the smart home control spectrum is not only navigated during setup or steady-state use but also actively re-navigated in moments of breakdown.

Table 4.6: **Taxonomy of user confusion in executing smart home functionality.**

Category	Description	Example Quote
Fragmented control (central vs. vendor)	Users unsure which interface (central vs. vendor) exposes functionality.	<i>“In the Nest app you can see the history... but not on Google Home.”</i> (P5)
Hidden / hard-to-find features	Poor discoverability of routines, shortcuts, or advanced options.	<i>“Every time I go to set a routine, I have to relearn it.”</i> (P3)
Missing or partial integrations	Features expected to exist but absent or incomplete.	<i>“I can stream the camera, but I can’t move it with voice.”</i> (P5)
Ecosystem lock-in & workarounds	Vendor ecosystems discourage cross-platform setups; workarounds reduce control.	<i>“They assume you’re Hue or IKEA exclusively.”</i> (P7)
Multi-user household challenges	Routines or settings tied to single accounts, limiting family use.	<i>“If I add a routine in the Tapo app, [my wife] can’t change it.”</i> (P5)

4.4.4 Discoverability Failures across the Control Spectrum (RQ2b)

Our interviews reveal that users frequently struggle to realize their intended goals with smart home systems, not because functionality is absent, but because its integration or accessibility—whether in terms of location, when it is available, or how it is surfaced—is unclear.

We identify five recurring categories of confusion and elaborate on each below.

Table 4.6 summarizes all these categories, including relevant quotes from interviewees.

1) Fragmented control between central and vendor apps. Participants’ mental models of control schemes often assumed that central hubs (e.g., Google Home, Alexa) could fully replicate vendor apps. When this expectation was violated, users had to switch between apps, leading to uncertainty about which scheme truly “owned” a functionality. E.g., Table 4.6—row 1 shows that Nest and Google Home have different features.

2) Hidden or hard-to-find features. Even when functionality existed, participants’ expectations of where it should reside did not align with the app design. Features like routines or shortcuts were buried in menus, clashing with the mental model that everyday actions should be easily accessible. Users who rarely engaged with a particular scheme were especially likely to misremember or misinterpret where to look, e.g., having to locate a feature all over again every time they need it (Table 4.6—row 2).

3) Missing or partial integrations. Participants often assumed that devices sharing a hub or assistant would integrate seamlessly. When integrations were absent or partial, like camera actuation features (Table 4.6–row 3), this violated their mental model of the hub as a universal coordinator, eroding confidence in central control.

4) Ecosystem lock-in and workarounds. Users expected that devices across ecosystems could be mixed and matched. When these expectations failed, e.g., accessing IKEA devices from Hue apps (Table 4.6–row 4), they turned to workarounds such as third-party apps. These choices reflect not just technical limitations but users’ mental models of what a “smart” home should enable.

5) Multi-user household challenges. Participants assumed that central routines or device automations would be shareable within a household. When vendor apps restricted access to a single account, this clashed with their model of a shared home environment, driving them to central hubs despite their own limitations. For instance, Tapo does not allow multiple household members (Table 4.6–row 5).

Across all categories, confusion arose less from the outright absence of functionality than from mismatches between participants’ mental models and system design. Users expected hubs to be universal, apps to be intuitive, integrations to be complete, and households to be collaborative. In practice, however, they lacked a clear taxonomy of where functionality resides, how it should be invoked, and whether it integrates with other devices. These findings show that confusion is not just an isolated annoyance but a systemic barrier that reshapes how users perceive and navigate smart home control schemes, influencing when they trust central control, fall back to vendor apps, or abandon features altogether.

4.5 Discussion

Our survey and interview results (Section 4.4) show that smart home owners *navigate across* layers of proxy-based control, from central, i.e., several proxies, to physical control, i.e., zero proxies. as they orchestrate intent, and that the spectrum of smart device control

plays a major role in user preferences and experiences, echoing the intuition by Davidoff et al. [186] and Brush et al. [246] that homes should support everyday experiences rather than mere device operation. We connect these findings to HCI and ubiquitous computing by reframing control as *transition* along two spectra (Figure 4.7): (1) the *locus of execution* (from centralized to local), which reflects where a device action might be initiated from, whether through a hub or on the physical device, and (2) the *exposure of the system* (from seamless to seamful), i.e., how users perceive device control, from having to set-and-forget to having fine-grained control of the devices. This *dual-spectrum* view helps explain under what conditions people keep infrastructures invisible and when they purposefully reveal seams, and it grounds actionable implications for hubs, vendor apps, and household governance.

These spectra are orthogonal: a person can operate centrally and hide seams for low-stakes routines, then shift downstream and reveal seams to accomplish a fine-grained adjustment that the hub cannot reach (e.g., voice can stream a camera feed, but pan, tilt, and zoom (PTZ) requires the vendor app). Our participants’ perspectives revealed the importance of designing for these transitions, rather than a single point. We expand on this below, with examples.

4.5.1 From device control to orchestrating transitions

In the consumer landscape, platform marketing emphasizes *seamlessness* and *invisible* infrastructure; for example, Matter aims to make multi-brand integration routine, promising an architecture that “just works” in daily use [213]. Users move freely along the spectrum, guided by situational factors such as perceived convenience, perceived reliability, and task.

While these narratives shape expectations, Bell and Dourish [247] argue that the seamless vision of ubiquitous computing remains less of a lived reality. In HCI, seamful design argues that exposing implementation gaps can be useful for diagnosis and

adaptation [248], [249], while infrastructure studies remind us that infrastructures are ordinarily taken for granted and become visible at breakdown, adoption, or scaling [250]. Managing and constructing a smart home space requires users to constantly weigh tradeoffs between convenience and control, automation and transparency, and speed and flexibility as discussed in Section 4.2.1.

We mapped participants’ interaction journey between physical and central control onto two orthogonal spectra that distinguished *where* individuals are operating and controlling their devices from, and *how* people are handling rough edges.

The locus of execution spectrum: Proxy-based control ↔ Physical control. The *locus of execution* refers to the level of abstraction at which smart home users control their devices [251], [252]: whether controlling them through a hub/ecosystem (central; shared scenes, routines, voice assistants) or invoking actions through the device-specific app (local; vendor app, on-device UI, physical interface). Higher-level proxy-based loci tend to provide *breadth* across devices and people; local loci provide *depth and assurance* via device-specific features, histories/logs, firmware access, and tangible fallbacks [152], [253].

Movement across loci is common. In our data, shifts toward physical control were precipitated by capability ceilings at the hub, device-specific needs (e.g., camera PTZ control or schedule histories), governance constraints (account scoping, co-editing), and lifecycle events (router or Internet service provider swaps, device replacement). For instance, participants with larger installations described relying on a hub for routine scenes yet still opening brand apps to adjust advanced parameters or to locate firmware updates (Section 4.4.2). These participants treated proxy-based control as a default broadcast channel for tasks, but reserved physical or local controls as authoritative sources about specific device states or capabilities.

The seam practice spectrum: Seam hiding ↔ Seam revealing. In an infrastructure such as a smart home, *seams* are the imperfections and complexities of systems [254].

While the locus of execution spectrum captures *where* users act in the infrastructure, the

seam practice spectrum captures *how* they choose to manage the visibility of that infrastructure as they move across layers. Rather than treating seams as a fixed property of the system, our data show seam practice as an ongoing, situated accomplishment: participants actively decide when to keep control flows invisible and when to expose the underlying infrastructure to get leverage. This everyday seam hiding or seamlessness reduces cognitive and social overhead (e.g., making it easier for other household members to just talk to the assistant rather than learning multiple apps), and resonates with prior critiques of frictionless interfaces [255], [256].

When smart device expectations failed, participants engaged in deliberate seam revealing processes that went beyond what prior seamful design examples [158], [257]. Participants systematically peeled back layers. They start at a high-level proxy (hub or voice), then open vendor apps to inspect device states, history, or logs, and finally manipulate device-local controls or power to test hypotheses about what was failing (Figure 4.6b). By *appropriating* seams as diagnostic handles, validating prior work describing how technical exposures serve as resources for sensemaking during breakdown [258], [259]. Our dual-spectrum lens (Figure 4.7) frames **seam hiding as the default when proxy-based locus of execution is stable, whereas seam revealing is a manual intervention for users to regain precision, understanding, or accountability when that locus fails them.**

Transitions as design. By creating this dual spectrum, we notice that movement across control layers should be preserved rather than eliminated; this gives individuals agency to trade off convenience against fine-grained control, accountability, and recovery. Rather than arguing the control scheme for “hub vs. per-brand” claims, the agency for participants to move across the spectrum is a less explored design space where participants adopted *central + invisible* configurations for repetitive, low-stakes tasks; they shifted toward *physical + seamful* configurations for fine-grained control, accountability, or assurance. This perspective also addresses timeliness critiques: even as standards evolve, households will

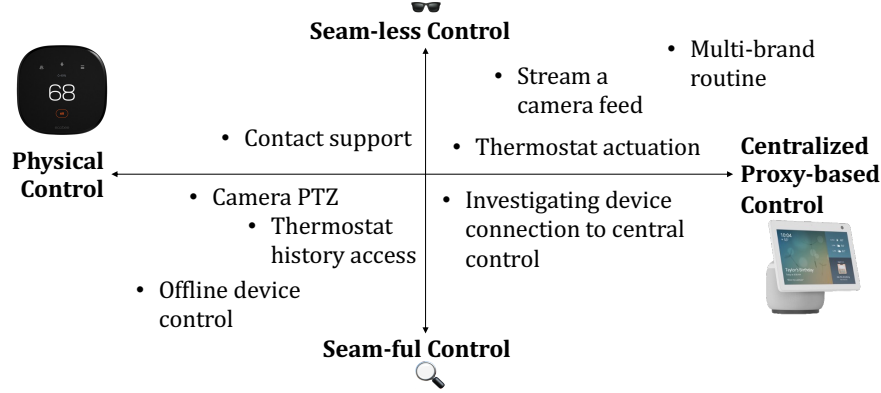


Figure 4.7: **Dual smart home control spectrum.** *The proxy-based vs physical control axis refers to the locus of execution spectrum, whereas the seam-less vs seam-ful axis refers to the seam practice spectrum. The user can navigate the locus spectrum based on their needs, whereas the vendors give limited navigability of the seam spectrum, almost enforcing how users will experience their control interfaces’ integration.*

continue to encounter breakdowns, maintenance, mixed accounts, and perceived risk—conditions that *require* traversing axes.

4.5.2 Smart homes as information ecologies of control

Our work reveals that when a user considers adding or modifying devices in their home, or even interacting with their home, it is not just the devices, but the modes of control that matter. The IoT/smart home device marketplace typically frames value in terms of devices with “more” or “smarter” features [260], [261], [262]. However, our findings show that the ways those devices can be controlled are equally important for how people evaluate and live with smart home devices over time. Put more technically, smart homes are often described as a single coherent system [263], [264], [265]; however, our findings suggest a different framing, where smart homes are thought of as information ecologies of control. While traditional information ecology theory focuses broadly on human-technology configurations [266], [267], participant experiences show that the loci of execution also function as a significant part of this ecology. Households cultivate combinations of these “species,” or groups of devices and associated control modalities (such as P5, who had different loci depending on the room of their house), and adjust them as their setup,

routine, and environment change.

Movements along the locus of execution spectrum are less about isolated interface preferences and more about ecological preference. Adding a new device, changing homes or internet providers, experiencing repeated failures, or other external stimuli prompted participants to re-balance their control ecology. This included disabling a routine (P3), abandoning a hub/device (P5), or falling back to physical device control (P8). Furthermore, troubleshooting was revealed as a moment where not only a malfunctioning device was examined, but the entire smart home ecology.

Participants continuously evaluated their smart home ecology in terms of its usefulness and ability to integrate into their day-to-day lives. This evaluation helped participants identify which loci of execution they were willing to depend on. By framing smart homes as evolving information ecologies, which include loci of execution, we shift attention from evaluating individual products to understanding how they participate in and reshape household ecologies. Next, we translate this lens into concrete design implications and open questions for future smart home systems.

4.5.3 Design implications and open questions

A dual-spectrum stance shifts attention from choosing a “best” locus to *supporting transitions* to making crossings legible, low-friction, and accountable. Importantly, *control loci do not map one-to-one to physical devices*: the hardware could contain several loci (third-party app vs. second-party app vs. first-party physical control), and a routine state can be “hidden” while device state remains visible or vice versa.

Design hubs for transparency and ceilings. Centralized interfaces should disclose *capability ceilings* and provide explicit *bridges* to device-local depth. When a hub cannot reach feature/parameter-level control (e.g., camera PTZ, thermostat histories [268]), the UI should indicate this and provide a deep-link to the native panel. This reduces “hunt-the-setting” failures, lessens the mental burden of switching platforms and

remembering device names, and matches observed escalations in Section 4.4.2. Designing away these transitions in pursuit of a perfectly seamless hub risks trapping users at a single, brittle locus of execution with no graceful way to regain control when expectations are violated.

Provenance and interoperability legibility. When scenes (predefined multi-device routines) span mixed brands or partial standards support, displaying a pre-flight check with the “weakest link” highlighted and present guided degradation paths (e.g., “device offline—try local control”), along with *who executed what, where*. Provenance-forward feedback increases trust when hub-instantiated actions trigger device responses, helping smart home users construct mental models and accelerating troubleshooting paths we observed in Section 4.4.3. This approach prioritizes coverage over device-specific depth. This works well for routine, low-stakes state changes and whole-home coordination; when a capability ceiling or failure is detected, the pre-flight/provenance cues a *depth-first* pivot (e.g., open vendor app, use a physical control) to resolve the outlier. For example, centralized interfaces should be honest about what they cannot accomplish and point users to the right vendor/device-specific path when they hit those limits.

Toward a flexible, brand-agnostic integration layer. The fragmentation that drives the app-minimizing and seam-revealing behaviors we observed (Sections 4.4.1 and 4.4.2) is, at root, an interoperability problem—but a partial and shifting one rather than a fixed wall. Vendors expose a common subset through emerging standards like Matter [213] while reserving advanced, brand-defining features for their own apps, and what they expose evolves over time. The promising direction is therefore not a layer that assumes full cross-brand cooperation, but a *flexible* integration layer that adapts to whatever each vendor currently exposes: coordinating intent over the standardized subset, and gracefully bridging—via first-class deep-links and automation hooks—to vendor apps for the capabilities they reserve. This raises several research problems. First, coordinating automations across a *shifting* interoperable/proprietary boundary and re-planning as

capabilities migrate between the standard and vendor apps; this is where the interface layer meets the partial action observability of Rascal and the cross-source conflict-free coordination of CoMesh. Second, discovering and adapting to each vendor’s exposed capabilities at runtime—degrading gracefully and re-routing rather than failing when a feature is only reachable through a brand app. Third, making the current gaps *legible* so users can distinguish “not yet supported” from “reserved by the vendor,” factor this into purchase decisions, and move fluidly to wherever a capability actually lives (a role large language models could assist by mapping features to interfaces). Treating interoperability as an evolving negotiation—nudged by standards and, as the EU’s mandated common USB-C connector shows, sometimes by regulation—keeps such a layer aligned with this chapter’s central stance: it should lower the cost of operating centrally while still *exposing*, never hiding, the bridges that let users descend to device-local control for depth or assurance.

Governing households while respecting privacy. Proxy-based interfaces should treat routines and devices as shared resources with roles, co-editing, and change histories by default (Table 4.6). Concretely, this points toward role-based access control (RBAC) [269] for the home, where administrators (e.g., parents) retain full control while guests or children receive scoped, limited permissions—a consistent privacy model across control schemes rather than per-vendor, ad hoc settings. Revealing device states and last-known state offers tangible backstops for high-stakes actions, helping smart home users navigate across the two spectra. For example, prior HCI work shows users prefer simple, proactive controls in the home and benefit from tangible mechanisms for privacy and assurance [270], [271], [272]. This design echoes the delicate balance wherein tangible interfaces reveal how household members shift between central and local loci and between hiding and revealing seams. In practice, this requires deeper engagement with stakeholders in the smart home, especially those whose voices are traditionally unheard (children, secondary users, nannies, etc.), as current spectra navigating designs often hurt these users most [273], [274].

Working with such users and including them as an integral part of the design process helps establish both smart home devices and their corresponding control spectra as “sources of empowerment”, which can make moving between control methods in the home a less daunting experience [275].

Whole-home concerns, not per-device settings. While our study centered on control of individual devices and routines, users also hold concerns that span the home as a whole—energy conservation, safety, and shared expectations about how the household should behave. Prior work shows such cross-home concerns are both real and tractable at that level: Knox [276] fortifies smart spaces with safety guarantees enforced across the space rather than device by device, and SafeHome [277] makes the case for visibility and atomicity over the home as a unit. Complementarily, Zaidi et al. [182] find that *human needs*—rather than device semantics—drive how users want conflicts between routines resolved, and thus what automation behavior they prefer in the first place. Taken together, these point toward platforms that let occupants express home-level goals and values once and have them shape behavior across every locus of execution, instead of requiring them to be re-encoded per device. Goal-oriented LLM-based systems such as Sasha [278], which translate loosely specified user goals into concrete device actions, suggest one promising substrate for doing so.

AI assistance for debugging and proactive recommendations. The troubleshooting journeys we observed (Section 4.4.3) are largely user-driven: occupants must notice a failure, hypothesize a cause, and walk the control spectrum themselves to test it. AI assistants could shoulder much of this burden—inspecting routine and device state to localize a fault, explaining it in plain language, and recommending the concrete next step (e.g., which locus of execution to descend to). Beyond reactive debugging, systems could make *proactive* recommendations: flagging routines likely to conflict or silently fail, and suggesting transitions before a breakdown forces one. Such assistance must remain legible and optional, augmenting rather than replacing the reflective control practices our

participants valued.

Open Questions: Evaluating Transition Quality. Our work opens up two further directions on transition quality. First, there is a need to characterize what information and granularity different user roles (and levels of expertise) need at the moment of crossing to be effective without overload. Second, there is a need to develop evaluation metrics for transition quality: e.g., time to successful handoff; rate of stranded actions; clarity of ceilings and provenance; and recovery burden after lifecycle events (Section 4.4.1).

4.6 Limitations and Future Work

Our methodology used surveys and interviews to elicit responses from participants regarding their past experiences and current perceptions.

Future research should look to expand on this work through studies that focus on clear categories of smart home owners, based on characteristics such as location (e.g., temperate zones vs. frigid vs. coastal), size of home (apartment vs. single-family home vs. mansion), and other factors. In addition, our survey respondents and interview participants likely skewed more towards a tech-savvy population, especially given that we recruited using Prolific [279]. Thus, future work can also explore perceptions and experiences of users who may be overall less aware of smart home device capabilities. Relatedly, future studies should differentiate between generations of users, who may hold different expectations of control, adopt different troubleshooting strategies, and prefer different points on the control spectrum. Older adults in particular have distinct needs: prior work built and deployed a personal assistance system that delivered medication and appointment reminders alongside wearable health sensors for independent and assisted living [280], and the accompanying field study found that the social structures surrounding older users shape both what such a system must do and whether it is accepted at all [281]. Because the needs and wants of each population—and indeed each country—differ substantially, these are hard and open problems that require specialized study of the human *and* the

systems aspects together. This thesis provides groundwork for both: the control spectrum characterized here, and the abstractions that support movement along it, give such studies a common vocabulary and a set of mechanisms to build on. In addition, surveys and interviews rely on participants to reflect on their interactions with smart home devices, rather than having the research team observe these interactions directly. Future work can utilize more observational methodologies and ethnographic studies to expand our results by providing additional context on how participants control their devices.

Our results are tied to the technology mix used during data collection. However, as technology evolves and matures, and new standards such as Matter [213] emerge, future work can incorporate our broader implications and findings into its design. Interaction patterns related to user transitions across the control spectrum will remain even as such standards develop, which will require future research to use our approach to understand *how* users' interaction patterns change with these standards in place.

Our work focused primarily on a single user in a smart home; while participants did discuss how other users in the home influenced their perceptions and preferences, our study elicited only one opinion from each smart home. Previous work has described smart homes as more sociotechnical systems, often containing multiple users with diverse experiences, desires, and perceptions [149]. Future work should more explicitly consider the sociotechnical nature by conducting studies with multiple users in the same home, examining how control schemes and methods in a single household affect different users in a smart home, and building upon the perspectives elicited in this work.

4.7 Conclusion

We present Control in Context, a mixed-methods study of how and why smart home users navigate the spectrum from physical per-device interaction to proxy-based central control. Through a survey of $N = 60$ smart home owners and semi-structured follow-up interviews with $N = 8$ participants, we find that smart home control is a dynamic, reflective, and

often labor-intensive process rather than a static preference. Users continuously adapt their control strategies—and their device procurement decisions—as they encounter tradeoffs between convenience, reliability, precision, and cognitive load. In particular, over half of participants reported switching control schemes based on task context, and nearly all interview participants described at least one device or platform abandoned after a negative troubleshooting experience. Our work emphasizes that *future systems should not only support multiple control schemes, but also actively guide users’ movement across them*. Researchers should similarly reframe their evaluation metrics and system assumptions to reflect the temporal and spatial heterogeneity of real-world smart home usage. This work is an important initial step towards home management systems that are truly user-centric rather than device-centric.

Chapter 5 Conclusion and Future Work

5.1 Summary of Contributions

This thesis has advanced a single vision—designing the control of smart spaces (environments instrumented with IoT devices for automated sensing and actuation) around the needs of the people who inhabit them—across three *independent* planes of the smart-space stack: the *device action plane*, the *smart-space automation plane*, and the *human interaction plane*. The three contributions are unified by this vision, not by a flow of insight from one to the next; together they make smart-space control decentralized, observable, and grounded in how real occupants actually live with their devices.

1. **The device action plane—RASC (Chapter 2)** introduces a new abstraction for IoT device control that closes the gap between the fire-and-forget RPC primitive and the rich action-progress information that users and developers need. RASC exposes four lifecycle states—Request, Acknowledgment, Start, and Complete—and its implementation (Rascal) adds an adaptive polling strategy and DAG-aware scheduling layer atop existing IoT platforms without requiring device modifications. Evaluation on real smart-home automation workloads shows that RASC detects action completion within 2–16 seconds in over 90% of cases, and that its scheduling policies outperform state-of-the-art baselines by 10–55%.
2. **The smart-space automation plane—CoMesh (Chapter 3)** presents the first fully decentralized, hub-less control plane for Sense-Trigger-Actuate (STA) operations in large commercial edge IoT meshes. By organizing devices into self-managing coterie that use zero-message exchange, quorum-based agreement, epoch-based migration, and locality-sensitive hashing, CoMesh eliminates the single-hub bottleneck and cloud dependency that limit today’s deployments. Formal analysis and Raspberry Pi experiments confirm its safety, liveness, scalability, and fault tolerance. CoMesh demonstrates that resilient, edge-first control of large smart

spaces is not only theoretically sound but practically deployable.

3. **The human interaction plane—Control in Context (Chapter 4)** provides the first empirical characterization of how smart-home users navigate the full spectrum of control schemes—from physical per-device interaction to proxy-based centralized hubs—in the context of everyday tasks, device failures, and evolving home setups. Through a survey of $N = 60$ smart-home owners and semi-structured interviews with $N = 8$ participants, this work reveals that control is a fluid, reflective process guided by situational factors rather than ecosystem boundaries. The findings expose the mismatch between platform design assumptions and real user behavior, and provide grounded design principles for future systems that treat the user—not the vendor ecosystem—as the center of design.

Collectively, this work turns three planes of smart-space control—the device action plane (how actions are observed and programmed), the smart-space automation plane (where control logic runs), and the human interaction plane (how occupants interact with it)—from sources of fragility and frustration into design targets shaped by the needs of the people who inhabit the space.

5.2 Future Research Directions

Each contribution opens new research directions; we outline the most promising below, several of which we are actively pursuing.

5.2.1 Richer Observability and Controllability (RASC++)

RASC opens the black-box RPC into a *gray-box* abstraction with visibility at the Request, Acknowledgment, Start, and Complete points; several extensions promise still richer control. *Group-based RASC* would generalize the lifecycle to commands issued to *groups* of devices (e.g., “all lights”), with group-level progress and consistency semantics. Moving

from gray-box toward *white-box* observability would surface per-action and per-routine progress—progress-bar-style indicators—for near-full visibility into long-running commands. *Controllability* is a third gap: a routine today runs uninterruptibly once fired, whereas users and automations should be able to safely *modify or interrupt* in-flight routines. Together these enable a RASC-powered *debugging assistant* that records action timelines, pinpoints the first divergence from expected behavior, and proposes remediation—directly addressing the troubleshooting pain points and the call for AI-assisted debugging that Control in Context raised (Section 4.5.3)—alongside mechanisms to specify and assure *safety clauses* across concurrent automations. A further direction is scaling RASC to deployments with many concurrent users, where the same devices are polled on behalf of many actions at once. Because lifecycle states are per-device rather than per-user, two complementary mechanisms apply: *batching* status queries so that one poll serves every routine currently interested in a device, and *caching*, where a recent poll response is reused to answer subsequent requests instead of issuing a fresh one. Both keep polling overhead proportional to the number of active devices rather than the number of users, and—importantly—help stay within the rate limits that device APIs impose.

5.2.2 RASC as a General Distributed Abstraction

RASC’s core insight—that long-running distributed actions have meaningful intermediate states callers should observe—applies well beyond IoT. RPCs, microservice invocations, and workflow steps share the same request/acknowledge/start/complete structure, so RASC and its DAG-aware scheduling could serve as a general alternative to fire-and-forget RPC in distributed databases, cloud workflow orchestration, and robotics pipelines.

5.2.3 Toward User-Centered Smart-Home Platforms

Control in Context’s findings open a research agenda for the next generation of smart-home platforms (“home operating systems”)—systems designed around how people

actually exercise control rather than around vendor ecosystems. Promising directions include: (i) surfacing *capability ceilings* and providing principled *bridges* to device-local controls when a hub cannot reach a feature; (ii) making *provenance and interoperability transparent*—attributing who or what executed an action and running pre-flight “weakest-link” checks before multi-brand routines; (iii) a *brand-agnostic integration layer* that adapts to whatever each vendor exposes rather than assuming full cooperation; (iv) treating routines and devices as *shared resources*, with role-based access and change histories for multi-user homes; (v) explicitly *supporting transitions* along the control spectrum rather than locking users into a single scheme; (vi) *hierarchical control domains* for large multi-tenant buildings. An apartment building resembles an office building—many users, many uses—and this thesis’s contributions carry over to it directly: RASC’s action observability and the control-spectrum framing hold just as well at building scale as in a single home, and CoMesh is designed for precisely such large deployments. What such buildings add is *hierarchy*: a superintendent may set control policy for shared corridors, HVAC, and entrances, while each tenant retains authority over their own unit, with the two composing predictably. Formalizing these nested control domains—who may observe and actuate what, and how policies at one level constrain another—is a natural extension of the abstractions developed here; and (vii) adapting to the *competencies* of users—calibrating how much control and complexity an interface exposes to each occupant’s current expertise, and growing with them as that expertise develops (echoing Control in Context’s observation that users expand their setups as their skills mature). A complementary open question is a principled framework for *when* a deployment should decentralize its control plane, given device count, churn, latency, and privacy constraints. More than our own next step, we see this as a call to the systems and HCI communities to build smart-space operating systems that explicitly address these concerns.

5.2.4 LLM-Assisted and Declarative Automation Authoring

As discussed in the introduction, the rise of LLMs largely does not alter this thesis’s conclusions: RASC’s abstraction and CoMesh’s decentralized design stand independently of how automations are authored, and for these planes LLMs act as accelerators—easing the specification of causal dependencies in RASC and simplifying routine authoring atop CoMesh. The human interaction plane deserves more nuance. As LLM assistants become part of how occupants program and troubleshoot their homes, they may reshape the very practices Control in Context observed—an LLM conversation becomes a new locus of execution—or proxy, possibly adding another layer to the control spectrum—and a new step in troubleshooting journeys—so revisiting the study as adoption grows, alongside building the user-facing assistance Control in Context calls for, is itself a direction this thesis opens. A similar question arises for new interaction hardware: should head-worn devices such as smart glasses displace smartphones, gaze and vision could make the physical device itself the entry point to the control spectrum—overlaying a device’s state and the routines that touch it for in-situ troubleshooting—while the growing presence of *transient* devices that move in and out of a space turns the choice between cloud control (continuity across spaces) and local control (data stays within the space) into a tradeoff platforms should expose for users to set. RASC’s structured action model is a natural substrate for higher-level authoring. Large language models could synthesize automations from natural-language intent, grounded in RASC’s *fallback actions* and *co-actions* so that generated routines degrade gracefully rather than failing silently—e.g., “if the garage door fails to close, send an alert and lock the side door instead.” Complementarily, *declarative interfaces* for specifying and tracking routines—stating desired end-states and invariants rather than imperative step sequences—would let users and tools reason about, verify, and maintain automations as homes evolve.

5.2.5 Coordination for Mobile and Robotic Smart Spaces

Smart spaces are becoming mobile: large factories increasingly deploy humanoid and wheeled robots that work between stations and roam the floor, and homes host a growing fleet of mobile devices (vacuums, lawn mowers, telepresence robots). CoMesh already tolerates such movement *reactively*—its coterie-migration machinery treats a robot that moves as churn and re-forms coterie around its new location. A richer direction is *proactive* coterie management: predicting robot motion from task schedules, floor-plan structure, or learned trajectories, and reshaping coterie *before* a robot arrives, so that the sensing and actuation it depends on are already coordinated locally. Other domains already exploit predictability this way to good effect: satellite systems schedule downlinks and bifurcate queries around known orbital passes [282], [283], and short-form video delivery caches content ahead of anticipated demand [284]. Combined with RASC-style lifecycle observability for long-running robot actions, this would extend the thesis’s decentralized, human-centered control stack to spaces whose members are themselves in motion.

5.2.6 Language-Model Inference at the Smart-Space Edge

Capable small language models now run on commodity hardware, enabling on-device natural-language interaction without the cloud. Building on RASC’s observability and CoMesh’s edge-native infrastructure, future work could distribute language-model inference across the heterogeneous, often-idle devices already present in a smart space [12]—adaptively choosing pipeline, tensor, or mixed-mode placement to fit tight memory and wireless budgets—for low-latency natural-language control that survives cloud or Internet service provider outages.

References

- [1] IoT Analytics, *State of IoT 2025: Number of connected IoT devices growing 14% to 21.1 billion globally*, <https://iot-analytics.com/number-connected-iot-devices/>, 2025.
- [2] C. Kelly, *EE: Average UK smart home will have 50 connected devices by 2023*, <https://www.totaltele.com/500103/EE-Average-UK-SmartHome-will-have-50-connected-devices-by-2023>, Accessed: October 2022, 2018.
- [3] D. Vasisht, Z. Kapetanovic, J. Won, X. Jin, R. Chandra, S. Sinha, A. Kapoor, M. Sudarshan, and S. Stratman, “FarmBeats: An IoT platform for Data-Driven agriculture,” in *USENIX Symposium on Networked Systems Design and Implementation (NSDI)*, 2017.
- [4] G. Lampropoulos, S. K., and T. Anastasiadis, “Internet of things in the context of industry 4.0: An overview,” *International Journal of Entrepreneurial Knowledge (IJEK)*, 2019. DOI: 10.37335/ijek.v7i1.84
- [5] Memoori, *IoT devices in Smart Commercial Buildings 2025 to 2030*, <https://memoori.com/portfolio/iot-devices-smart-commercial-buildings-2025/>, 2025.
- [6] Memoori, *The Global Market for the Internet of Things in Smart Commercial Buildings*, <https://memoori.com/portfolio/internet-of-things-smart-buildings-2023/>, 2022.
- [7] A. N. Sivakumar, S. Modi, M. V. Gasparino, C. Ellis, A. E. B. Velasquez, G. Chowdhary, and S. Gupta, “Learned visual navigation for under-canopy agricultural robots,” in *Robotics: Science and Systems*, 2021. DOI: 10.15607/rss.2021.xvii.019

- [8] Home Assistant, *Hundred percent local home assistant (no cloud)*, <https://community.home-assistant.io/t/100-local-home-assistant-no-cloud/455682>, 2022.
- [9] Connectivity Standards Alliance, *Build With Matter: Smart Home Device Solution*, <https://csa-iot.org/all-solutions/matter/>, May 2025.
- [10] M. Grothaus, *The perils of a cloud-connected smart home*. Jun. 2019. [Online]. Available: <https://www.fastcompany.com/90358396/that-major-google-outage-meant-some-nest-users-couldnt-unlock-doors-or-use-the-ac>
- [11] Azure, *Azure IoT Edge pricing*, <https://azure.microsoft.com/en-us/pricing/details/iot-edge#pricing>.
- [12] D. Wei, J. Zhang, H. Li, R. Han, Z. Xue, Y. Peng, X. Pang, Y. Ma, and J. Li, “HyperEdge: An edge CDN infrastructure for cost efficient video streaming,” in *23rd USENIX Symposium on Networked Systems Design and Implementation (NSDI 26)*, Renton, WA: USENIX Association, May 2026, pp. 2635–2650, ISBN: 978-1-939133-54-0.
- [13] A. D. Birrell and B. J. Nelson, “Implementing remote procedure calls,” *ACM Transactions on Computer Systems*, vol. 2, no. 1, pp. 39–59, 1984. DOI: 10.1145/2080.357392
- [14] A. Karanika, K.-S. Wang, H.-T. Liang, S. Sundram, and I. Gupta, “RASC: Enhancing observability & programmability in smart spaces,” in *23rd USENIX Symposium on Networked Systems Design and Implementation (NSDI 26)*, USENIX Association, 2026, pp. 237–252.
- [15] A. Karanika, R. Yang, X. Ma, J. Wang, S. Sundram, and I. Gupta, “There is more control in egalitarian edge IoT meshes,” *IEEE Transactions on Network and Service Management*, vol. 23, pp. 896–909, 2026. DOI: 10.1109/TNSM.2025.3608796

- [16] A. Zaidi, A. Karanika, T.-C. Cheng, Y.-S. Chiang, C. Cobb, I. Gupta, and K. Karahalios, “Control in context: How smart home users navigate proxy-based schemes,” in *Proceedings of the 2026 CHI Conference on Human Factors in Computing Systems*, ser. CHI '26, ACM, 2026. DOI: 10.1145/3772318.3790549
- [17] K. Akpınar, T. Ballard, K. A. Hua, K. Li, S. Tarnpradab, and J. Ye, “COMMIT: A multimedia collaboration system for future workplaces with the internet of things,” in *MMSys*, 2017.
- [18] L. Turchet, C. Fischione, G. Essl, D. Keller, and M. Barthet, “Internet of musical things: Vision and challenges,” *IEEE Access*, vol. 6, 2018. DOI: 10.1109/access.2018.2872625
- [19] J. Kent, *The smart home in 2025: Outlook and opportunities*, <https://www.parksassociates.com/blogs/in-the-news/the-smart-home-in-2025-outlook-and-opportunities>, Jan. 2025.
- [20] G. Home, *Google Home API: Supported starters, conditions, and actions*, <https://developers.home.google.com/automations/starters-conditions-and-actions>.
- [21] Home Assistant, *Integrations*, <https://www.home-assistant.io/integrations/>.
- [22] IFTTT - Automate business & home, <https://ifttt.com/>.
- [23] Node-RED, <https://nodered.org/>.
- [24] *Comment on "what do you use ifttt for?"* https://www.reddit.com/r/homeautomation/comments/li3shd/comment/gn13kg4/?utm_source=share&utm_medium=web3x&utm_name=web3xcss&utm_term=1&utm_content=share_button, Accessed: October 2024, 2021. [Online]. Available: https://www.reddit.com/r/homeautomation/comments/li3shd/comment/gn13kg4/?utm_source=share&utm_medium=web3x&utm_name=web3xcss&utm_term=1&utm_content=share_button
- [25] Google, *gRPC codebase*, <https://github.com/grpc/grpc>, 2025.

- [26] Google, *gRPC: A high-performance, open source universal RPC framework*, <https://grpc.io/>, 2025.
- [27] M. Slee, A. Agarwal, and M. Kwiatkowski, “Thrift: Scalable cross-language services implementation,” in *USENIX Symposium on Operating Systems Design and Implementation (OSDI)*, 2007. [Online]. Available: <https://thrift.apache.org/static/files/thrift-20070401.pdf>
- [28] The Apache Thrift Developers, *Apache Thrift: A scalable cross-language RPC framework*, <https://github.com/apache/thrift>, 2025.
- [29] *Amazon Alexa*, <https://developer.amazon.com/alexa>.
- [30] *Google Home*, https://store.google.com/us/product/google_home.
- [31] C. Cornelio and M. Diab, “RECOVER: A neuro-symbolic framework for failure detection and recovery,” in *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS) (2024)*, 2024. DOI: 10.1109/iros58592.2024.10801853
- [32] Y. Zhang, P. Tokmakov, M. Hebert, and C. Schmid, “A structured model for action detection,” in *IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, Jun. 2019. DOI: 10.1109/cvpr.2019.01021
- [33] A. Quinn, J. Flinn, and M. Cafarella, “You can’t debug what you can’t see: Expanding observability with the OmniTable,” in *USENIX Workshop on Hot Topics in Operating Systems (HotOS)*, 2019, pp. 163–169.
- [34] J. Levin and T. A. Benson, “Viperprobe: Rethinking microservice observability with eBPF,” in *IEEE International Conference on Cloud Networking (CloudNet)*, IEEE, 2020, pp. 1–8. DOI: 10.1109/cloudnet51028.2020.9335808
- [35] O. Mart, C. Negru, F. Pop, and A. Castiglione, “Observability in Kubernetes cluster: Automatic anomalies detection using Prometheus,” in *HPCC/SmartCity/DSS*, IEEE, 2020, pp. 565–570. DOI: 10.1109/hpcc-smartcity-dss50907.2020.00071

- [36] T. Nelson, D. Yu, Y. Li, R. Fonseca, and S. Krishnamurthi, “Simon: Scriptable interactive monitoring for SDNs,” in *ACM Special Interest Group on Data Communication (SIGCOMM)*, 2015, pp. 1–7.
- [37] P. Santhoshini, K. H. Prabha, S. Jaibalaji, C. K. Vishwa, H. R. Kumar, and S. M. Sundar, “Implementation of automatic lawn mower system for smart lawn maintenance,” in *2024 4th International Conference on Artificial Intelligence and Signal Processing (AISP)*, IEEE, 2024, pp. 01–07. DOI: 10.1109/aisp61711.2024.10870614
- [38] K. M. Tsui, M. Desai, H. A. Yanco, and C. Uhlik, “Exploring use cases for telepresence robots,” in *ACM/IEEE International Conference on Human-Robot Interaction (HRI)*, 2011, pp. 11–18. DOI: 10.1145/1957656.1957664
- [39] *Home Assistant*, <https://www.home-assistant.io/>.
- [40] *Home Assistant: Classifying the internet of things*, <https://www.home-assistant.io/blog/2016/02/12/classifying-the-internet-of-things/>, 2016.
- [41] HP Support, *Fix hp print job stuck in queue*, <https://support.hp.com/us-en/help/printer/print-job-stuck>, HP Inc.
- [42] Nuki Developers Forum, *Error “423 Locked” implies lock is busy and cannot accept another action*, <https://developer.nuki.io/t/error-423-locked-and-general-error-500/30457>, 2024.
- [43] Home Assistant Community Forum User Vorta, *Hue bridge rate-limits state change events to 1 per second per device*, <https://community.home-assistant.io/t/philips-hue-switch-delay/342266>, 2022.
- [44] Protolabs, *Cool idea award winner: SmartCan*, <https://www.protolabs.com/about-us/cool-idea-award/smartcan/>.

- [45] S. B. Ahsan, R. Yang, S. A. Noghabi, and I. Gupta, “Home, SafeHome: Smart home reliability with visibility and atomicity,” in *European Conference on Computer Systems (EuroSys)*, 2021.
- [46] C. Shen, K. Ramamritham, and J. A. Stankovic, “Resource reclaiming in multiprocessor real-time systems,” *IEEE Transactions on Parallel and Distributed Systems*, vol. 4, no. 4, pp. 382–397, 1993. DOI: 10.1109/71.219754
- [47] G. Manimaran, C. S. R. Murthy, M. Vijay, and K. Ramamritham, “New algorithms for resource reclaiming from precedence constrained tasks in multiprocessor real-time systems,” *Journal of Parallel and Distributed Computing*, vol. 44, no. 2, pp. 123–132, 1997. DOI: 10.1006/jpdc.1997.1354
- [48] I. Gupta, G. Manimaran, and C. S. R. Murthy, “A new strategy for improving the effectiveness of resource reclaiming algorithms in multiprocessor real-time systems,” *Journal of Parallel and Distributed Computing*, vol. 60, no. 1, pp. 113–133, 2000. DOI: 10.1006/jpdc.1999.1594
- [49] A. S. Tanenbaum and A. S. Woodhull, *Operating Systems: Design and Implementation*, 3rd. Upper Saddle River, NJ: Prentice Hall, 2006, See Chapter 2, Section 2.4.2: Shortest Job First, pp. 82–83, ISBN: 0-13-142938-8.
- [50] *HomeAssistantCore*, <https://github.com/home-assistant/core>, 2024.
- [51] *Virtual devices for Home Assistant*, <https://github.com/twrecked/hass-virtual>, 2023.
- [52] *IoTBench test-suite*, <https://github.com/IoTBench/IoTBench-test-suite/tree/master/openHAB>.
- [53] *SmartThings open-source DeviceTypeHandlers and SmartApps code*, <https://github.com/SmartThingsCommunity/SmartThingsPublic/tree/master/smartapps>.

- [54] F. C. for Sustainable Energy Systems, *Multifamily programmable thermostat data*, <https://catalog.data.gov/dataset/multifamily-programmable-thermostat-data>, 2022. DOI: 10.25984/1844177
- [55] Y. E. Ioannidis and V. Poosala, “Balancing histogram optimality and practicality for query result size estimation,” *ACM SIGMOD Record*, vol. 24, no. 2, pp. 233–244, 1995. DOI: 10.1145/568271.223841
- [56] L. N. Vaserstein, “Markov processes over denumerable products of spaces, describing large systems of automata,” *Problems of Information Transmission*, vol. 5, no. 3, pp. 47–52, 1969.
- [57] L. V. Kantorovich, “On the transfer of masses,” *Doklady Akademii Nauk SSSR*, vol. 37, no. 7-8, pp. 227–229, 1942.
- [58] TIBCO Software Inc., *Project Flogo: Open source event-driven ecosystem*, <https://www.flogo.io>.
- [59] ROS Wiki, *Actionlib*, <http://wiki.ros.org/actionlib>, 2026.
- [60] gRPC Authors, *gRPC Basics - Go: Server-side streaming RPC*, <https://grpc.io/docs/languages/go/basics#server-side-streaming-rpc>, 2026.
- [61] OASIS Standard, *MQTT Version 5.0*, <https://docs.oasis-open.org/mqtt/mqtt/v5.0/mqtt-v5.0.html>, 2019.
- [62] M. Roman, C. Hess, R. Cerqueira, A. Ranganathan, R. Campbell, and K. Nahrstedt, “A middleware infrastructure for active spaces,” *IEEE Pervasive Computing*, vol. 1, no. 4, pp. 74–83, 2002. DOI: 10.1109/MPRV.2002.1158281
- [63] P. A. Vicaire, E. Hoque, Z. Xie, and J. A. Stankovic, “Bundle: A group based programming abstraction for cyber physical systems,” in *ICCPs*, 2010, pp. 32–41.

- [64] S. Fu and S. Ratnasamy, “dSpace: Composable abstractions for smart spaces,” in *ACM Symposium on Operating Systems Principles (SOSP)*, 2021, pp. 295–310.
- [65] 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)*, 2014. DOI: 10.1109/iccps.2014.6843717
- [66] A. Sengupta, T. Leesatapornwongsa, M. S. Ardekani, and C. A. Stuardo, “Transactuations: Where transactions meet the physical world,” in *USENIX Annual Technical Conference (ATC)*, 2019.
- [67] M. Ardekani, R. Singh, N. Agrawal, D. Terry, and R. Suminto, “Rivulet: A fault-tolerant platform for smart-home applications,” in *ACM/IFIP/USENIX International Middleware Conference (Middleware)*, 2017.
- [68] M. Norris, Z. B. Celik, P. Venkatesh, S. Zhao, P. McDaniel, A. Sivasubramaniam, and G. Tan, “IoTRepair: Systematically addressing device faults in commodity IoT,” in *ACM/IEEE Conference on Internet of Things Design and Implementation (IoTDI)*, 2020, pp. 142–148. DOI: 10.1109/iotdi49375.2020.00021
- [69] E. Anceaume, G. Cabillic, P. Chevochot, and I. Puant, “HADES: A middleware support for distributed safety-critical real-time applications,” in *IEEE International Conference on Distributed Computing Systems (ICDCS)*, 1998, pp. 344–351.
- [70] M. Kashef, Y. Liu, K. Montgomery, and R. Candell, “Wireless cyber-physical system performance evaluation through a graph database approach,” *Journal of computing and information science in engineering*, vol. 21, no. 2, p. 021 009, 2021.
- [71] D. Cardinaels, R. Ramakers, T. Veuskens, T. Pietrzak, G. A. Roveló Ruiz, and K. Luyten, “Every move you make: Visualizing near-future motion under delay for telerobotics,” in *Proceedings of the 2026 CHI Conference on Human Factors in Computing Systems*, ser. CHI ’26, New York, NY, USA: Association for Computing

- Machinery, 2026. DOI: 10.1145/3772318.3791452 [Online]. Available: <https://doi.org/10.1145/3772318.3791452>
- [72] G. Banavar, T. Chandra, B. Mukherjee, J. Nagarajao, R. E. Strom, and D. C. Sturman, “An efficient multicast protocol for content-based publish-subscribe systems,” in *IEEE International Conference on Distributed Computing Systems (ICDCS)*, IEEE, 1999, pp. 262–272.
 - [73] A. Carzaniga, D. S. Rosenblum, and A. L. Wolf, “Design and evaluation of a wide-area event notification service,” *ACM Transactions on Computer Systems*, vol. 19, no. 3, pp. 332–383, 2001. DOI: 10.1145/380749.380767
 - [74] M. K. Aguilera, R. E. Strom, D. C. Sturman, M. Astley, and T. D. Chandra, “Matching events in a content-based subscription system,” in *ACM Symposium on Principles of Distributed Computing (PODC)*, 1999, pp. 53–61. DOI: 10.1145/301308.301326
 - [75] B. Segall and D. Arnold, “Elvin has left the building: A publish/subscribe notification service with quenching,” *AUUG*, pp. 243–255, 1997.
 - [76] D. W. Scott, “On optimal and data-based histograms,” *Biometrika*, vol. 66, no. 3, pp. 605–610, 1979. DOI: 10.1093/biomet/66.3.605
 - [77] D. Freedman and P. Diaconis, “On the histogram as a density estimator: L 2 theory,” *Zeitschrift für Wahrscheinlichkeitstheorie und verwandte Gebiete*, vol. 57, no. 4, pp. 453–476, 1981. DOI: 10.1007/bf01025868
 - [78] K. H. Knuth, “Optimal data-based binning for histograms,” *arXiv preprint physics/0605197*, 2006.
 - [79] J. D. Scargle, J. P. Norris, B. Jackson, and J. Chiang, “Studies in astronomical time series analysis. VI. bayesian block representations,” *The Astrophysical Journal*, vol. 764, no. 2, p. 167, 2013. DOI: 10.1088/0004-637x/764/2/167

- [80] H. V. Jagadish, N. Koudas, S. Muthukrishnan, V. Poosala, K. C. Sevcik, and T. Suel, "Optimal histograms with quality guarantees," in *International Conference on Very Large Data Bases (VLDB)*, vol. 98, 1998, pp. 24–27.
- [81] *Samsung SmartThings*, <https://www.smartthings.com/>.
- [82] D. Liu, T. F. Abdelzaher, T. Wang, Y. Hu, J. Li, S. Liu, M. Caesar, D. Kalasapura, J. Bhattacharyya, N. Srour, J. Kim, G. Wang, G. Kimberly, and S. Yao, "IoBT-OS: Optimizing the sensing-to-decision loop for the internet of battlefield things," in *2022 International Conference on Computer Communications and Networks (ICCCN)*, 2022. DOI: 10.1109/icccn54977.2022.9868920
- [83] Z. Fatima, M. H. Tanveer, S. Zardari, L. F. Naz, H. Khadim, N. Ahmed, and M. Tahir, "Production plant and warehouse automation with iot and industry 5.0," *Applied Sciences*, vol. 12, no. 4, 2022. DOI: 10.3390/app12042053
- [84] E. Research, *Internet of things in agriculture market: Forecast to 2030*, 2022. [Online]. Available: <https://www.emergenresearch.com/industry-report/iot-in-agriculture-market>
- [85] X. Guo, L. Shangguan, Y. He, N. Jing, J. Zhang, H. Jiang, and Y. Liu, "Saiyan: Design and implementation of a low-power demodulator for LoRa backscatter systems," in *USENIX Symposium on Networked Systems Design and Implementation (NSDI)*, 2022.
- [86] J. Song, S. Han, A. Mok, D. Chen, M. Lucas, M. Nixon, and W. Pratt, "WirelessHART: Applying wireless technology in real-time industrial process control," in *IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS)*, 2008. DOI: 10.1109/rtas.2008.15
- [87] ISO Central Secretary, Tech. Rep. ISO/IEC TR 29110-1:2016, 2016. [Online]. Available: <https://www.iso.org/standard/62711.html>

- [88] I. F. Akyildiz and X. Wang, “A survey on wireless mesh networks,” *IEEE Communications Magazine*, vol. 43, no. 9, S23–S30, 2005. DOI: 10.1109/MCOM.2005.1509968
- [89] J. Lynch, Z. Liu, C. Li, M. Ghobadi, and H. Balakrishnan, “The case for decentralized fallback networks,” in *ACM Workshop on Hot Topics in Networks (HotNets)*, 2024. DOI: 10.1145/3696348.3696883
- [90] S. B. Ahsan, R. Yang, S. A. Noghabi, and I. Gupta, “Home, SafeHome: Smart home reliability with visibility and atomicity,” in *European Conference on Computer Systems (EuroSys)*, 2021.
- [91] Thingsquare, *Cloudless IoT: IoT without the cloud*, <https://www.thingsquare.com/blog/articles/iot-without-the-cloud/>, 2021.
- [92] S. J. Moore, C. D. Nugent, S. Zhang, and I. Cleland, “Iot reliability: A review leading to 5 key research directions,” *CCF Transactions on Pervasive Computing and Interaction*, vol. 2, 2020. DOI: 10.1007/s42486-020-00037-z
- [93] C. Sengul, M. J. Miller, and I. Gupta, “Adaptive probability-based broadcast forwarding in energy-saving sensor networks,” *ACM Transactions on Sensor Networks*, vol. 4, no. 2, 2008. DOI: 10.1145/1340771.1340772
- [94] OLSR.org, *Optimized link state routing protocol*, https://en.wikipedia.org/wiki/Optimized_Link_State_Routing_Protocol.
- [95] J. Dunagan, N. J. A. Harvey, M. B. Jones, D. Kostić, M. Theimer, and A. Wolman, “FUSE: Lightweight guaranteed distributed failure notification,” in *USENIX Symposium on Operating Systems Design and Implementation (OSDI)*, 2004.
- [96] R. Yang, J. Wang, J. Hu, S. Zhu, Y. Li, and I. Gupta, “Medley: A membership service for IoT networks,” *IEEE Transactions on Network and Service Management*, vol. 19, no. 3, 2022. DOI: 10.1109/tnsm.2022.3196268

- [97] G. Han, H. Xu, T. Q. Duong, J. Jiang, and T. Hara, "Localization algorithms of wireless sensor networks: A survey," *Telecommunication Systems*, vol. 52, 2013. DOI: 10.5121/ijcnc.2013.5603
- [98] I. Stoica, R. Morris, D. Karger, M. F. Kaashoek, and H. Balakrishnan, "Chord: A scalable peer-to-peer lookup service for internet applications," in *ACM Special Interest Group on Data Communication (SIGCOMM)*, 2001.
- [99] G. DeCandia, D. Hastorun, M. Jampani, G. Kakulapati, A. Lakshman, A. Pilchin, S. Sivasubramanian, P. Voshall, and W. Vogels, "Dynamo: Amazon's highly available key-value store," in *ACM Symposium on Operating Systems Principles (SOSP)*, 2007.
- [100] G. I. Apecechea, *Cryptographic sortition in blockchains: The importance of vrfs*, 2019. [Online]. Available: <https://soracom.io/uptime-reliability-blog/>
- [101] H. Garcia-Molina, "Elections in a distributed computing system," *IEEE Transactions on Computers*, vol. C-31, no. 1, 1982. DOI: 10.1109/tc.1982.1675885
- [102] K. Krishna and M. Narasimha Murty, "Genetic k-means algorithm," *IEEE Trans. on Systems, Man, and Cybernetics, Part B (Cybernetics)*, vol. 29, no. 3, 1999. DOI: 10.1109/3477.764879
- [103] M. Datar, N. Immorlica, P. Indyk, and V. S. Mirrokni, "Locality-sensitive hashing scheme based on p-stable distributions," in *SoCG04: Annual ACM Symposium on Computational Geometry 2004*, 2004. DOI: 10.1145/997817.997857
- [104] B. Charyyev and M. H. Gunes, "Iot traffic flow identification using locality sensitive hashes," in *ICC 2020 - 2020 IEEE International Conference on Communications (ICC)*, 2020. DOI: 10.1109/icc40277.2020.9148743
- [105] N. Giatrakos, A. Deligiannakis, M. Garofalakis, and Y. Kotidis, "Omnibus outlier detection in sensor networks using windowed locality sensitive hashing," *Future*

- Generation Computer Systems*, vol. 110, 2020. DOI: 10.1016/j.future.2018.04.046
- [106] V. M. Zolotarev, *One-dimensional stable distributions* (Translations of Mathematical Monographs). Amer. Math. Soc., 1986, vol. 65. DOI: 10.1090/mmono/065
 - [107] G. F. Riley and T. R. Henderson, “The ns-3 network simulator.,” in *Modeling and Tools for Network Simulation*, 2010. DOI: 10.1007/978-3-642-12331-3_2
 - [108] C. McNally, *Google nest wi-fi review 2022*, 2022. [Online]. Available: <https://www.reviews.org/internet-service/google-nest-wifi-review/>
 - [109] A. Karanika, R. Yang, X. Ma, J. Wang, S. Sundram, and I. Gupta, *CoMesh: Fully-decentralized control for sense-trigger-actuate routines in edge meshes*, <https://arxiv.org/abs/2303.00207>, 2023. arXiv: 2303.00207 [cs.DC].
 - [110] J. Segarra, *How soracom has maintained 99.99% uptime for the past 3 years*, <https://soracom.io/uptime-reliability-blog/>, Jan. 2024.
 - [111] C. Dixon, R. Mahajan, S. Agarwal, A. J. Brush, B. Lee, S. Saroiu, and P. Bahl, “An operating system for the home,” in *USENIX Symposium on Networked Systems Design and Implementation (NSDI)*, 2012.
 - [112] P. H. Su, C.-S. Shih, J. Y.-J. Hsu, K.-J. Lin, and Y.-C. Wang, “Decentralized fault tolerance mechanism for intelligent IoT/M2M middleware,” in *2014 IEEE World Forum on Internet of Things (WF-IoT)*, 2014. DOI: 10.1109/wf-iot.2014.6803115
 - [113] K. Fall, “A delay-tolerant network architecture for challenged internets,” in *ACM Special Interest Group on Data Communication (SIGCOMM)*, 2003. DOI: 10.1145/863955.863960
 - [114] I. Gupta, R. Van Renesse, and K. P. Birman, “A probabilistically correct leader election protocol for large groups,” in *International Symposium on Distributed Computing (DISC)*, 2000. DOI: 10.1007/3-540-40026-5_6

- [115] S. Lai, B. Ravindran, and H. Cho, “Heterogenous quorum-based wake-up scheduling in wireless sensor networks,” *IEEE Transactions on Computers*, vol. 59, no. 11, 2010. DOI: 10.1109/tc.2010.20
- [116] I. Stojmenovic, D. Liu, and X. Jia, “A scalable quorum-based location service in ad hoc and sensor networks,” *International Journal of Communication Networks and Distributed Systems (IJCNDs)*, vol. 1, no. 1, 2008. DOI: 10.1504/ijcnds.2008.017205
- [117] V. Poirot, B. Al Nahas, and O. Landsiedel, “Paxos made wireless: Consensus in the air,” in *International Conference on Embedded Wireless Systems and Networks (EWSN)*, 2019.
- [118] M. Xu, C. Liu, Y. Zou, F. Zhao, J. Yu, and X. Cheng, “wChain: A fast fault-tolerant blockchain protocol for multihop wireless networks,” *IEEE Transactions on Wireless Communications*, 2021. DOI: 10.1109/twc.2021.3078639
- [119] U. Khalid, M. Asim, T. Baker, P. C. Hung, M. A. Tariq, and L. Rafferty, “A decentralized lightweight blockchain-based authentication mechanism for iot systems,” *Cluster Computing*, vol. 23, no. 3, 2020. DOI: 10.1007/s10586-020-03058-6
- [120] K. Yeow, A. Gani, R. W. Ahmad, J. J. Rodrigues, and K. Ko, “Decentralized consensus for edge-centric internet of things: A review, taxonomy, and research issues,” *IEEE Access*, vol. 6, 2017.
- [121] R. Xu, Y. Chen, E. Blasch, and G. Chen, “Blendcac: A smart contract enabled decentralized capability-based access control mechanism for the iot,” *Computers*, vol. 7, no. 3, 2018. DOI: 10.20944/preprints201805.0079.v1
- [122] Z. Khayyat, K. Awara, A. Alonazi, H. Jamjoom, D. Williams, and P. Kalnis, “Mizan: A system for dynamic load balancing in large-scale graph processing,” in *European Conference on Computer Systems (EuroSys)*, 2013.

- [123] K. He, E. Rozner, K. Agarwal, W. Felter, J. Carter, and A. Akella, “Presto: Edge-based load balancing for fast datacenter networks,” in *ACM Special Interest Group on Data Communication (SIGCOMM)*, 2015.
- [124] J. Li, D. Cordes, and J. Zhang, “Power-aware routing protocols in ad hoc wireless networks,” *IEEE Wireless Communications*, vol. 12, no. 6, 2005. DOI: 10.1109/mwc.2005.1561947
- [125] Z. Nezami, K. Zamanifar, K. Djemame, and E. Pournaras, “Decentralized edge-to-cloud load balancing: Service placement for the internet of things,” *IEEE Access*, vol. 9, 2021. DOI: 10.1109/access.2021.3074962
- [126] Y. Sahni, J. Cao, S. Zhang, and L. Yang, “Edge Mesh: A new paradigm to enable distributed intelligence in internet of things,” *IEEE Access*, vol. 5, 2017. DOI: 10.1109/access.2017.2739804
- [127] A. Fuerst and P. Sharma, “Locality-aware load-balancing for serverless clusters,” in *ACM International Symposium on High-Performance Parallel and Distributed Computing (HPDC) ('22)*, 2022. DOI: 10.1145/3502181.3531459
- [128] Thread Group, *Thread*. [Online]. Available: <https://threadgroup.org/>
- [129] A. Mehra, *Smart home market worth \$151.4 billion by 2024*, <https://www.marketsandmarkets.com/PressReleases/global-smart-homes-market.asp>, 2020.
- [130] Statista, *Smart home – united states*, <https://www.statista.com/outlook/cmo/smart-home/united-states>, Accessed: June 2026.
- [131] A. Tripathi, N. Sindhvani, R. Anand, and A. Dahiya, “Role of iot in smart homes and smart cities: Challenges, benefits, and applications,” in *IoT-based Smart Applications*, Springer, 2022, pp. 199–217. DOI: 10.1007/978-3-031-04524-0_12
- [132] *Ecobee app*, <https://www.ecobee.com/en-us/smart-home/>.

- [133] *Phllips hue app*,
<https://www.philips-hue.com/en-in/explore-hue/apps/philips-hue-app>.
- [134] GE Cync, *GE Cync smart home devices*,
<https://www.gelighting.com/smart-home>, Accessed: 2024.
- [135] Apple, *Apple smart home*,
<https://www.apple.com/shop/smart-home/accessories>, Accessed: 2024.
- [136] Honeywell, *Honeywell home smart devices*,
<https://www.honeywellhome.com/us/en/>, Accessed: 2024.
- [137] Siemens, *Siemens home automation*, <https://www.siemens.com/global/en/products/automation/systems/industrial/plc/logo/home-automation.html>, Accessed: 2024.
- [138] Samsung, *Samsung SmartThings*, <https://www.samsung.com/us/smartthings/>, Accessed: 2024.
- [139] R. K. Kodali, S. Soratkal, and L. Boppana, “Iot-based control of appliances,” in *2016 International Conference on Computing, Communication and Automation (ICCCA)*, IEEE, 2016, pp. 1293–1297.
- [140] M. A. Khan, I. Ahmad, A. N. Nordin, A. Ahmed, H. Mewada, Y. Daradkeh, S. Rasheed, E. T. Eldin, and M. Shafiq, “Smart android based home automation system using internet of things (iot),” *Sustainability*, 2022. DOI: 10.3390/su141710717
- [141] M.-A. Kaufhold, T. Riebe, M. Bayer, and C. Reuter, “‘we do not have the capacity to monitor all media’: A design case study on cyber situational awareness in computer emergency response teams,” in *Proceedings of the 2024 CHI Conference on Human Factors in Computing Systems*, 2024, pp. 1–16. DOI: 10.1145/3613904.3642368
- [142] *Apple homekit*, <https://www.apple.com/home-app/>.

- [143] Google, *Create & manage routines in Google Home*,
<https://support.google.com/googlenest/answer/7029585>, Accessed: June 2026, 2024.
- [144] Amazon, *Create an Alexa routine in the Alexa app*, <https://www.amazon.com/gp/help/customer/display.html?nodeId=G2PYLKJN3XVZ55EQ>, Accessed: June 2026, 2025.
- [145] B. O’Boyle, *What are Google Assistant routines and how to do multiple actions with one command?*
<https://www.pocket-lint.com/smart-home/news/google/143928-google-assistant-routines-how-to-do-multiple-actions-with-one-command>,
 Accessed: January 2022, 2021.
- [146] A. K. Sikder, L. Babun, Z. B. Celik, H. Aksu, P. McDaniel, E. Kirda, and A. S. Uluagac, “Who’s controlling my device? multi-user multi-device-aware access control system for shared smart home environment,” *ACM Transactions on Internet of Things*, vol. 3, no. 4, pp. 1–39, 2022. DOI: 10.1145/3543513
- [147] Y. Yao, L. Huang, Y. He, Z. Ma, X. Xu, and H. Mi, “Reviewing and reflecting on smart home research from the human-centered perspective,” in *Proceedings of the 2023 CHI Conference on Human Factors in Computing Systems*, ser. CHI ’23, Hamburg, Germany: Association for Computing Machinery, 2023, ISBN: 9781450394215. DOI: 10.1145/3544548.3580842 [Online]. Available: <https://doi.org/10.1145/3544548.3580842>
- [148] G. Zimmermann and G. Vanderheiden, “The universal control hub: An open platform for remote user interfaces in the digital home,” in *International Conference on Human-Computer Interaction*, Springer, 2007, pp. 1040–1049.

- [149] C. Geeng and F. Roesner, “Who’s in control? interactions in multi-user smart homes,” in *Proceedings of the 2019 CHI Conference on Human Factors in Computing Systems*, 2019, pp. 1–13.
- [150] J. Nichols and B. A. Myers, “Studying the use of handhelds to control smart appliances,” in *23rd International Conference on Distributed Computing Systems Workshops*, IEEE, 2003, pp. 274–279.
- [151] M. Alaa, A. A. Zaidan, B. B. Zaidan, M. Talal, and M. L. M. Kiah, “A review of smart home applications based on internet of things,” *Journal of Network and Computer Applications*, vol. 97, pp. 48–65, 2017. DOI: 10.1016/j.jnca.2017.08.017
- [152] F. Corno, L. De Russis, and A. Monge Roffarello, “From users’ intentions to if-then rules in the internet of things,” *2021 ACM Transactions on Information Systems (TOIS)*, vol. 39, no. 4, pp. 1–33, 2021. DOI: 10.1145/3447264
- [153] T. Perumal, S. K. Datta, and C. Bonnet, “Iot device management framework for smart home scenarios,” in *2015 IEEE 4Th Global Conference on Consumer Electronics (GCCE)*, IEEE, 2015, pp. 54–55. DOI: 10.1109/gcce.2015.7398711
- [154] O. Salman, I. Elhajj, A. Kayssi, and A. Chehab, “An architecture for the internet of things with decentralized data and centralized control,” in *2015 IEEE/ACS 12th International Conference of Computer Systems and Applications (AICCSA)*, IEEE, 2015, pp. 1–8. DOI: 10.1109/aiccsa.2015.7507265
- [155] J. D. Day and H. Zimmermann, “The osi reference model,” *Proceedings of the IEEE*, vol. 71, no. 12, pp. 1334–1340, 1983. DOI: 10.1109/proc.1983.12775
- [156] M. Duque, S. Pink, M. Mortimer, Y. Strengers, R. Martin, L. Nicholls, B. Horan, A. Eugene, and S. Thomson, “Troubleshooting as a method in covid-19 times: Smart home ethnographies and remote aged care innovation,” *Digital Creativity*, vol. 33, no. 3, pp. 188–203, 2022. DOI: 10.1080/14626268.2022.2088560

- [157] M. R. Alam, M. B. I. Reaz, and M. A. M. Ali, “A review of smart homes—past, present, and future,” *IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews)*, vol. 42, no. 6, pp. 1190–1203, 2012. DOI: 10.1109/tsmcc.2012.2189204
- [158] S. Kuznetsov, A. S. Taylor, T. Regan, N. Villar, and E. Paulos, “At the seams: Diybio and opportunities for hci,” in *Proceedings of the Designing Interactive Systems Conference*, 2012, pp. 258–267.
- [159] H. F. Teigen, “Troubleshooting the connected home: Exploring the perspectives of non-initiators,” *Convergence: The International Journal of Research into New Media Technologies*, p. 13 548 565 241 268 062, 2024. DOI: 10.1177/13548565241268062
- [160] W. Wulf, E. Cohen, W. Corwin, A. Jones, R. Levin, C. Pierson, and F. Pollack, “Hydra: The kernel of a multiprocessor operating system,” *Communications of the ACM*, vol. 17, no. 6, pp. 337–345, 1974.
- [161] D. B. Golub, R. W. Dean, A. Forin, and R. F. Rashid, “Unix as an application program,” in *UsENIX Summer*, 1990, pp. 87–95.
- [162] D. M. Ritchie and K. Thompson, “The unix time-sharing system,” *Communications of the ACM*, vol. 17, no. 7, pp. 365–375, 1974. DOI: 10.1145/361011.361061
- [163] The Open Group, *Unix*, <https://www.opengroup.org/unix>.
- [164] *Linux kernel*, <https://www.kernel.org/>.
- [165] The FreeBSD Project, *Freebsd*, <https://www.freebsd.org/>.
- [166] *Macos*, <https://www.apple.com/macOS/>.
- [167] M. Tirmazi, A. Barker, N. Deng, M. E. Haque, Z. G. Qin, S. Hand, M. Harchol-Balter, and J. Wilkes, “Borg: The next generation,” in *Proceedings of the Fifteenth European Conference on Computer Systems*, 2020, pp. 1–14.

- [168] A. Verma, L. Pedrosa, M. Korupolu, D. Oppenheimer, E. Tune, and J. Wilkes, “Large-scale cluster management at google with borg,” in *Proceedings of the Tenth European Conference on Computer Systems*, 2015, pp. 1–17. DOI: 10.1145/2741948.2741964
- [169] M. Schwarzkopf, A. Konwinski, M. Abd-El-Malek, and J. Wilkes, “Omega: Flexible, scalable schedulers for large compute clusters,” in *Proceedings of the 8Th ACM European Conference on Computer Systems*, 2013, pp. 351–364.
- [170] L. Hernández, F. Kurniawan, and R. Alfred, “A wireless infrastructure prototype based on a cloud networking solution to optimize service provision in a health provider entity,” in *The Spirit of Recovery*, CRC Press, pp. 99–112. DOI: 10.1201/9781003331674-8
- [171] A. Lara, A. Kolasani, and B. Ramamurthy, “Network innovation using openflow: A survey,” *IEEE Communications Surveys & Tutorials*, vol. 16, no. 1, pp. 493–512, 2013.
- [172] B. Ur, M. Pak Yong Ho, S. Brawner, J. Lee, S. Mennicken, N. Picard, D. Schulze, and M. L. Littman, “Trigger-action programming in the wild: An analysis of 200,000 ifttt recipes,” in *Proceedings of the 2016 CHI Conference on Human Factors in Computing Systems*, 2016, pp. 3227–3231.
- [173] J. Gabhane, S. Thakare, and M. Craig, “Smart homes system using internet-of-things: Issues, solutions and recent research directions,” *International Research Journal of Engineering and Technology*, vol. 4, no. 5, pp. 1965–1969, 2017.
- [174] W. Brackenbury, A. Deora, J. Ritchey, J. Vallee, W. He, G. Wang, M. L. Littman, and B. Ur, “How users interpret bugs in trigger-action programming,” in *Proceedings of the 2019 CHI Conference on Human Factors in Computing Systems*, 2019, pp. 1–12. DOI: 10.1145/3290605.3300782

- [175] M. P. Woźniak, M. Koelle, S. Vöge, H. Müller, R. Krüger, and S. Boll, “Inhabiting interconnected spaces: How users shape and appropriate their smart home ecosystems,” in *Proceedings of the 2023 CHI Conference on Human Factors in Computing Systems*, 2023, pp. 1–18. DOI: 10.1145/3544548.3581497
- [176] T. OConnor, R. Mohamed, M. Miettinen, W. Enck, B. Reaves, and A.-R. Sadeghi, “Homesnitch: Behavior transparency and control for smart home iot devices,” in *Proceedings of the 12Th Conference on Security and Privacy in Wireless and Mobile Networks*, 2019, pp. 128–138.
- [177] A. Acar, H. Fereidooni, T. Abera, A. K. Sikder, M. Miettinen, H. Aksu, M. Conti, A.-R. Sadeghi, and S. Uluagac, “Peek-a-boo: I see your smart home activities, even encrypted!” In *Proceedings of the 13Th ACM Conference on Security and Privacy in Wireless and Mobile Networks*, 2020, pp. 207–218.
- [178] T. Hargreaves and C. Wilson, “Introduction: Smart homes and their users,” in *Smart Homes and Their Users*, Springer, 2017, pp. 1–14. DOI: 10.1007/978-3-319-68018-7_1
- [179] D. Buil-Gil, S. Kemp, S. Kuenzel, L. Coventry, S. Zakhary, D. Tilley, and J. Nicholson, “The digital harms of smart home devices: A systematic literature review,” *Computers in Human Behavior*, vol. 145, p. 107 770, 2023. DOI: 10.1016/j.chb.2023.107770
- [180] V. Koshy, J. S. S. Park, T.-C. Cheng, and K. Karahalios, ““We just use what they give us”: Understanding passenger user perspectives in smart homes,” in *Proceedings of the 2021 CHI Conference on Human Factors in Computing Systems*, 2021, pp. 1–14. DOI: 10.1145/3411764.3445598
- [181] Y.-S. Chiang, R.-C. Chang, Y.-L. Chuang, S.-Y. Chou, H.-P. Lee, I.-J. Lin, J.-H. Jiang Chen, and Y.-J. Chang, “Exploring the design space of user-system communication for smart home routine assistants,” in *Proceedings of the 2020 CHI*

- Conference on Human Factors in Computing Systems*, 2020, pp. 1–14. DOI: 10.1145/3313831.3376501
- [182] A. Zaidi, R. Yang, V. Koshy, C. Cobb, I. Gupta, and K. Karahalios, “A user-centric evaluation of smart home resolution approaches for conflicts between routines,” *Proceedings of the ACM on Interactive, Mobile, Wearable and Ubiquitous Technologies*, vol. 7, no. 1, pp. 1–35, 2023. DOI: 10.1145/3581997
- [183] A. Coskun, G. Kaner, and İ. Bostan, “Is smart home a necessity or a fantasy for the mainstream user? a study on users’ expectations of smart household appliances,” *International Journal of Design*, vol. 12, no. 1, pp. 7–20, 2018.
- [184] E. Kristen Mattson, *App-dependent or app-enabled?* <https://drkmattson.wordpress.com/2015/02/02/app-dependent-or-app-enabled/>, Jun. 2016.
- [185] N. Meng-Schneider, R. Yasa Kostas, K. Vaniea, and M. K. Wolters, “Multi-user smart speakers-a narrative review of concerns and problematic interactions,” in *Extended Abstracts of the 2023 CHI Conference on Human Factors in Computing Systems*, 2023, pp. 1–7. DOI: 10.1145/3544549.3585689
- [186] S. Davidoff, M. K. Lee, C. Yiu, J. Zimmerman, and A. K. Dey, “Principles of smart home control,” in *International Conference on Ubiquitous Computing*, Springer, 2006, pp. 19–34. DOI: 10.1007/11853565_2
- [187] S. FakhrHosseini, C. Lee, S.-H. Lee, and J. Coughlin, “A taxonomy of home automation: Expert perspectives on the future of smarter homes,” *Information Systems Frontiers*, pp. 1–18, 2024. DOI: 10.1007/s10796-024-10496-9
- [188] A. Alshehri and R. Sandhu, “Access control models for cloud-enabled internet of things: A proposed architecture and research agenda,” in *2016 IEEE 2Nd International Conference on Collaboration and Internet Computing (CIC)*, IEEE, 2016, pp. 530–538. DOI: 10.1109/cic.2016.081

- [189] E. Anthi, S. Ahmad, O. Rana, G. Theodorakopoulos, and P. Burnap, “Eclipseiot: A secure and adaptive hub for the internet of things,” *Computers & Security*, vol. 78, pp. 477–490, 2018. DOI: 10.1016/j.cose.2018.07.016
- [190] S. Basu and E. G. Sirer, “Trustless iot: A logic-driven architecture for iot hubs,” in *3Rd USENIX Workshop on Hot Topics in Edge Computing (HotEdge 20)*, 2020.
- [191] S. R. Devi, R. K. R, S. G. Priya, G. Sathi, S. N. Kumar, and D. M, “Design and development of a touch free smart home controlling system based on virtual reality (vr) technology,” in *2024 International Conference on Intelligent Systems for Cybersecurity (ISCS)*, 2024, pp. 1–6. DOI: 10.1109/ISCS61804.2024.10581017
- [192] R. Raj and N. Rai, “Voice controlled cyber-physical system for smart home,” in *Proceedings of the Workshop Program of the 19Th International Conference on Distributed Computing and Networking*, 2018, pp. 1–5. DOI: 10.1145/3170521.3170550
- [193] C. Shen, R. P. Singh, A. Phanishayee, A. Kansal, and R. Mahajan, “Beam: Ending monolithic applications for connected devices,” in *USENIX Annual Technical Conference (ATC)*, 2016.
- [194] A. Tavakoli, A. Kansal, and S. Nath, “On-line sensing task optimization for shared sensors,” in *ACM/IEEE International Conference on Information Processing in Sensor Networks (IPSN)*, 2010. DOI: 10.1145/1791212.1791219
- [195] K. Whitehouse, F. Zhao, and J. Liu, “Semantic streams: A framework for composable semantic interpretation of sensor data,” in *International Conference on Embedded Wireless Systems and Networks (EWSN)*, 2006. DOI: 10.1007/11669463_4
- [196] V. S. Gunge and P. S. Yalagi, “Smart home automation: A literature review,” *International Journal of Computer Applications*, vol. 975, no. 8887-8891, 2016.

- [197] I. Zavalysyn, A. Legay, A. Rath, and E. Rivière, “Sok: Privacy-enhancing smart home hubs,” *Proceedings on Privacy Enhancing Technologies*, 2022. DOI: 10.56553/popets-2022-0097
- [198] N. Apthorpe, D. Y. Huang, D. Reisman, A. Narayanan, and N. Feamster, “Keeping the smart home private with smart (er) iot traffic shaping,” *arXiv Preprint arXiv:1812.00955*, 2018.
- [199] B. Ur, E. McManus, M. Pak Yong Ho, and M. L. Littman, “Practical trigger-action programming in the smart home,” in *Proceedings of the 2014 CHI Conference on Human Factors in Computing Systems*, 2014, pp. 803–812. DOI: 10.1145/2556288.2557420
- [200] E. Zeng, S. Mare, and F. Roesner, “End user security and privacy concerns with smart homes,” in *Thirteenth Symposium on Usable Privacy and Security (SOUPS 2017)*, Santa Clara, CA: USENIX Association, Jul. 2017, pp. 65–80, ISBN: 978-1-931971-39-3. [Online]. Available: <https://www.usenix.org/conference/soups2017/technical-sessions/presentation/zeng>
- [201] P. Wang, “Connecting the parts with the whole: Toward an information ecology theory of digital innovation ecosystems,” *MIS Quarterly*, vol. 45, no. 1, pp. 397–422, 2021. DOI: 10.25300/misq/2021/15864
- [202] J. Huang and G. Xu, “Research on human-computer interaction framework model design of smart home terminal from the perspective of information ecology,” in *International Conference on Artificial Intelligence and Industrial Design (AIID 2022)*, SPIE, vol. 12612, 2023, pp. 106–117. DOI: 10.1117/12.2673091
- [203] C. Goumopoulos and A. Kameas, “Ambient ecologies in smart homes,” *The Computer Journal*, vol. 52, no. 8, pp. 922–937, 2009. DOI: 10.1093/comjnl/bxn042

- [204] F. D. Davis, R. P. Bagozzi, and P. R. Warshaw, "Technology acceptance model," *J Manag Sci*, vol. 35, no. 8, pp. 982–1003, 1989.
- [205] A. Shuhaiber and I. Mashal, "Understanding users' acceptance of smart homes," *Technology in Society*, vol. 58, p. 101 110, 2019. DOI: 10.1016/j.techsoc.2019.01.003
- [206] C. Zhou, Y. Qian, and J. Kaner, "A study on smart home use intention of elderly consumers based on technology acceptance models," *Plos One*, vol. 19, no. 3, e0300574, 2024. DOI: 10.1371/journal.pone.0300574
- [207] A. N. Tak, B. Becerik-Gerber, L. Soibelman, and G. Lucas, "A framework for investigating the acceptance of smart home technologies: Findings for residential smart hvac systems," *Building and Environment*, vol. 245, p. 110 935, 2023. DOI: 10.1016/j.buildenv.2023.110935
- [208] V. Kaptelinin and B. A. Nardi, *Activity Theory in HCI: Fundamentals and Reflections*. Morgan & Claypool Publishers, 2012, vol. 13.
- [209] A. Demeure, S. Caffiau, and J. Coutaz, "Activity based end-user-development for smart homes: Relevance and challenges," in *Workshop Proceedings of the 10Th International Conference on Intelligent Environments*, IOS Press, 2014, pp. 141–152. DOI: 10.3233/978-1-61499-411-4-141
- [210] E. Maier and U. Reimer, "Analysing the use of voice assistants in domestic settings through the lens of activity theory," in *International Conference on Human-Computer Interaction*, Springer, 2023, pp. 366–379. DOI: 10.1007/978-3-031-35602-5_26
- [211] H. Jin, G. Liu, D. Hwang, S. Kumar, Y. Agarwal, and J. I. Hong, "Peekaboo: A hub-based approach to enable transparency in data processing within smart homes," in *2022 IEEE Symposium on Security and Privacy (SP)*, IEEE, 2022, pp. 303–320. DOI: 10.1109/sp46214.2022.9833629

- [212] B. Lee, O. Kwon, I. Lee, and J. Kim, “Companionship with smart home devices: The impact of social connectedness and interaction types on perceived social support and companionship in smart homes,” *Computers in Human Behavior*, vol. 75, pp. 922–934, 2017. DOI: 10.1016/j.chb.2017.06.031
- [213] W. Zegeye, A. Jemal, and K. Kornegay, “Connected smart home over matter protocol,” in *2023 IEEE International Conference on Consumer Electronics (ICCE)*, 2023, pp. 1–7. DOI: 10.1109/ICCE56470.2023.10043520
- [214] Y. Shi and N. Liu, “Investigating control method of user interaction with multi-device in smart home,” in *International Conference on Human-Computer Interaction*, Springer, 2023, pp. 430–439. DOI: 10.1007/978-3-031-35939-2_32
- [215] M. Caporuscio, F. Flammini, N. Khakpour, P. Singh, and J. Thornadtsson, “Smart-troubleshooting connected devices: Concept, challenges and opportunities,” *Future Generation Computer Systems*, vol. 111, pp. 681–697, 2020. DOI: 10.1016/j.future.2019.09.004
- [216] R. Abdallah, L. Xu, and W. Shi, “Lessons and experiences of a diy smart home,” in *Proceedings of the Workshop on Smart Internet of Things*, 2017, pp. 1–6. DOI: 10.1145/3132479.3132488
- [217] W. He, J. Martinez, R. Padhi, L. Zhang, and B. Ur, “When smart devices are stupid: Negative experiences using home smart devices,” in *2019 IEEE Security and Privacy Workshops (SPW)*, IEEE, 2019, pp. 150–155. DOI: 10.1109/spw.2019.00036
- [218] M. Ansari, “Seamless security optimizing cybersecurity and design in smart home interiors,” in *Complexities and Challenges for Securing Digital Assets and Infrastructure*, IGI Global Scientific Publishing, 2025, pp. 47–72. DOI: 10.4018/979-8-3373-1370-2.ch003

- [219] J. E. Kim, G. Boulos, J. Yackovich, T. Barth, C. Beckel, and D. Mosse, “Seamless integration of heterogeneous devices and access control in smart homes,” in *2012 Eighth International Conference on Intelligent Environments*, IEEE, 2012, pp. 206–213. DOI: 10.1109/ie.2012.57
- [220] R. Garg and H. Cui, “Social contexts, agency, and conflicts: Exploring critical aspects of design for future smart home technologies,” *ACM Transactions on Computer-Human Interaction*, vol. 29, no. 2, pp. 1–30, 2022. DOI: 10.1145/3485058
- [221] C.-H. Lu, Y.-C. Lin, and L.-C. Fu, “Hide and not easy to seek: A hybrid weaving strategy for context-aware service provision in a smart home,” in *2008 IEEE Asia-Pacific Services Computing Conference*, IEEE, 2008, pp. 595–600. DOI: 10.1109/apscc.2008.77
- [222] C.-H. Lu and L.-C. Fu, “A human-centric and cooperative architecture for multiple target tracking in a smart home,” in *2009 Joint Conferences on Pervasive Computing (JCPC)*, IEEE, 2009, pp. 407–412. DOI: 10.1109/jcpc.2009.5420152
- [223] N. Balta-Ozkan, R. Davidson, M. Bicket, and L. Whitmarsh, “Social barriers to the adoption of smart homes,” *Energy Policy*, vol. 63, pp. 363–374, 2013. DOI: 10.1016/j.enpol.2013.08.043
- [224] I. Malik, A. Bhardwaj, H. Bhardwaj, and A. Sakalle, “Iot-enabled smart homes: Architecture, challenges, and issues,” *Revolutionizing Industrial Automation Through the Convergence of Artificial Intelligence and the Internet of Things*, pp. 160–176, 2023.
- [225] A. Müller, G. Miinz, and G. Carle, “Collecting router information for error diagnosis and troubleshooting in home networks,” in *2011 IEEE 36Th Conference on Local Computer Networks*, IEEE, 2011, pp. 764–769. DOI: 10.1109/lcn.2011.6115547
- [226] A. R. Alves, H. D. Moura, J. R. Borges, V. F. Mota, L. H. Cantelli, D. F. Macedo, and M. A. Vieira, “Homenetrescue: An sdn service for troubleshooting home

- networks,” in *NOMS 2018-2018 IEEE/IFIP Network Operations and Management Symposium*, IEEE, 2018, pp. 1–7. DOI: 10.1109/noms.2018.8406167
- [227] S. Palan and C. Schitter, “Prolific —a subject pool for online experiments,” *Journal of Behavioral and Experimental Finance*, vol. 17, pp. 22–27, 2018.
- [228] B. D. Douglas, P. J. Ewell, and M. Brauer, “Data quality in online human-subjects research: Comparisons between mturk, prolific, cloudresearch, qualtrics, and sona,” *PLOS ONE*, vol. 18, no. 3, 2023. DOI: 10.1371/journal.pone.0279720
- [229] P. Eyal, R. David, G. Andrew, E. Zak, and D. Ekaterina, “Data quality of platforms and panels for online behavioral research,” *Behavior Research Methods*, pp. 1–20, 2021. DOI: 10.3758/s13428-021-01694-3
- [230] M.-R. Kennedy, R. Huxtable, G. Birchley, J. Ives, and I. Craddock, ““a question of trust” and “a leap of faith”—study participants’ perspectives on consent, privacy, and trust in smart home research: Qualitative study,” *JMIR mHealth and uHealth*, vol. 9, no. 11, e25227, 2021. DOI: 10.2196/25227
- [231] G. Dermody, R. Fritz, C. Glass, M. Dunham, and L. Whitehead, “Factors influencing community-dwelling older adults’ readiness to adopt smart home technology: A qualitative exploratory study,” *Journal of Advanced Nursing*, vol. 77, no. 12, pp. 4847–4861, 2021. DOI: 10.1111/jan.14996/v1/review2
- [232] M. Hennink and B. N. Kaiser, “Sample sizes for saturation in qualitative research: A systematic review of empirical tests,” *Social Science & Medicine*, vol. 292, p. 114523, 2022. DOI: 10.1016/j.socscimed.2021.114523
- [233] E. Peer, D. Rothschild, and A. Gordon, “Platform over procedure: Online platforms that pre-vet participants yield higher data quality without sacrificing diversity,” *Working Paper*, 2023. DOI: 10.31234/osf.io/buzwn

- [234] J. D. Abbey and M. G. Meloy, “Attention by design: Using attention checks to detect inattentive respondents and improve data quality,” *Journal of Operations Management*, vol. 53, pp. 63–70, 2017. DOI: 10.1016/j.jom.2017.06.001
- [235] T. Lenzner, L. Kaczmirek, and A. Lenzner, “Cognitive burden of survey questions and response times: A psycholinguistic experiment,” *Applied Cognitive Psychology*, vol. 24, no. 7, pp. 1003–1020, 2010.
- [236] S. H. Khandkar, “Open coding,” *University of Calgary*, vol. 23, no. 2009, p. 2009, 2009.
- [237] N. M. Deterding and M. C. Waters, “Flexible coding of in-depth interviews: A twenty-first-century approach,” *Sociological Methods & Research*, vol. 50, no. 2, pp. 708–739, 2021.
- [238] K. A. Hallgren, “Computing inter-rater reliability for observational data: An overview and tutorial,” *Tutorials in Quantitative Methods for Psychology*, vol. 8, no. 1, p. 23, 2012. DOI: 10.20982/tqmp.08.1.p023
- [239] R. Falotico and P. Quatto, “Fleiss’ kappa statistic without paradoxes,” *Quality & Quantity*, vol. 49, pp. 463–470, 2015.
- [240] G. Dermody, R. Shibl, M. Wang, A. Ward, J. Watson, K. North, J. Blake, E. Mealy, A. M. Koay, S. Roomkham, P. L. Yeoh, and R. Fritz, “Multi perspective considerations for health smart home: Early phase exploratory study,” *Journal of Advanced Nursing*, vol. 82, no. 1, pp. 849–866, 2025. DOI: 10.1111/jan.16945
- [241] S. Zheng, N. Apthorpe, M. Chetty, and N. Feamster, “User perceptions of smart home iot privacy,” *Proceedings of the ACM on Human-computer Interaction*, vol. 2, no. CSCW, pp. 1–20, 2018. DOI: 10.1145/3274469
- [242] X. Wang, T. J. McGill, and J. E. Klobas, “I want it anyway: Consumer perceptions of smart home devices,” *Journal of Computer Information Systems*, 2020.

- [243] D. Singh, I. Psychoula, J. Kropf, S. Hanke, and A. Holzinger, “Users’ perceptions and attitudes towards smart home technologies,” in *Smart Homes and Health Telematics, Designing a Better Future: Urban Assisted Living: 16Th International Conference, ICOST 2018, Singapore, Singapore, July 10-12, 2018, Proceedings 16*, Springer, 2018, pp. 203–214. DOI: 10.1007/978-3-319-94523-1_18
- [244] A. Georgiev and S. Schlögl, “Smart home technology: An exploration of end user perceptions,” *Innovative Lösungen Für Eine Alternde Gesellschaft: Konferenzbeiträge Der SMARTER LIVES*, vol. 18, no. 20.02, p. 2018, 2018.
- [245] J. Wu, J. Chen, and W. Dou, “The internet of things and interaction style: The effect of smart interaction on brand attachment,” *Journal of Marketing Management*, vol. 33, no. 1-2, pp. 61–75, 2017.
- [246] A. B. Brush, B. Lee, R. Mahajan, S. Agarwal, S. Saroiu, and C. Dixon, “Home automation in the wild: Challenges and opportunities,” in *Proceedings of the 2011 CHI Conference on Human Factors in Computing Systems*, 2011, pp. 2115–2124.
- [247] G. Bell and P. Dourish, “Yesterday’s tomorrows: Notes on ubiquitous computing’s dominant vision,” *Personal and Ubiquitous Computing*, vol. 11, no. 2, pp. 133–143, 2007.
- [248] M. Chalmers, “Seamful Design and Ubicomp Infrastructure,” in *Proceedings Of Ubicomp 2003 Workshop at the Crossroads: The Interaction Of HCI And Systems Issues In Ubicomp*, 2003, pp. 577–584.
- [249] M. Chalmers, “Seamful design: Showing the seams in wearable computing,” in *IEE Eurowearable ’03*, vol. 2003, Birmingham, UK: IEE, 2003, pp. 11–16. DOI: 10.1049/ic:20030140 Accessed: Sep. 2, 2025.
- [250] S. L. Star and K. Ruhleder, “Steps towards an ecology of infrastructure: Complex problems in design and access for large-scale collaborative systems,” in *Proceedings of the 1994 ACM Conference on Computer Supported Cooperative Work*, ser. CSCW

- '94, Chapel Hill, North Carolina, USA: Association for Computing Machinery, 1994, pp. 253–264, ISBN: 0897916891. DOI: 10.1145/192844.193021 [Online]. Available: <https://doi.org/10.1145/192844.193021>
- [251] S. T. Perrault, E. Lecolinet, Y. P. Bourse, S. Zhao, and Y. Guiard, “Physical loci: Leveraging spatial, object and semantic memory for command selection,” in *Proceedings of the 2015 CHI Conference on Human Factors in Computing Systems*, 2015, pp. 299–308.
- [252] A. Rip and J. Schot, “Identifying loci for influencing the dynamics of technological development,” in *Shaping Technology. Guiding Policy; Concepts Spaces and Tools*, Edward Elgar, 2002, pp. 158–176. DOI: 10.4337/9781035352937.00012
- [253] X. Chen, S. Chen, Z. Jin, H. Bian, Z. Chen, and H. Li, “Expressing the needs in smart home: What is the end users’ favorite way,” *ACM Transactions on Computer-Human Interaction*, vol. 32, no. 2, pp. 1–38, 2025. DOI: 10.1145/3715114
- [254] S. Madakam and R. Ramaswamy, “Smart homes (conceptual views),” in *2014 2Nd International Symposium on Computational and Business Intelligence*, IEEE, 2014, pp. 63–66. DOI: 10.1109/iscbi.2014.21
- [255] N. Hengesbach, “Undoing seamlessness: Exploring seams for critical visualization,” in *CHI Conference on Human Factors in Computing Systems Extended Abstracts*, 2022, pp. 1–7. DOI: 10.1145/3491101.3519703
- [256] S. Inman and D. Ribes, “” beautiful seams” strategic revelations and concealments,” in *Proceedings of the 2019 CHI Conference on Human Factors in Computing Systems*, 2019, pp. 1–14.
- [257] M. Chalmers, A. Dieberger, K. Höök, and Å. Rudström, “Social navigation and seamful design,” *Cognitive Studies: Bulletin of the Japanese Cognitive Science Society*, vol. 11, no. 3, pp. 171–181, 2004.

- [258] E. Dao, A. Muresan, K. Hornbæk, and J. Knibbe, “Bad breakdowns, useful seams, and face slapping: Analysis of vr fails on youtube,” in *Proceedings of the 2021 Chi Conference on Human Factors in Computing Systems*, 2021, pp. 1–14. DOI: 10.1145/3411764.3445435
- [259] S. Benford, A. Crabtree, M. Flintham, A. Drozd, R. Anastasi, M. Paxton, N. Tandavanitj, M. Adams, and J. Row-Farr, “Can you see me now?” *ACM Transactions on Computer-Human Interaction (TOCHI)*, vol. 13, no. 1, pp. 100–133, 2006. DOI: 10.1145/1143518.1143522
- [260] K. Sun, J. Li, Y. Zou, J. Radesky, C. Brooks, and F. Schaub, “Unfulfilled promises of child safety and privacy: Portrayals and use of children in smart home marketing,” *Proceedings of the ACM on Human-Computer Interaction*, vol. 8, no. CSCW1, pp. 1–29, 2024. DOI: 10.1145/3637422
- [261] B. Basarir-Ozel, H. B. Turker, and V. A. Nasir, “Identifying the key drivers and barriers of smart home adoption: A thematic analysis from the business perspective,” *Sustainability*, vol. 14, no. 15, p. 9053, 2022. DOI: 10.3390/su14159053
- [262] M. Nasirinejad and S. Sampalli, “Evaluating consumer behavior, decision making, risks, and challenges for buying an iot product,” *IoT*, vol. 4, no. 2, pp. 78–94, 2023. DOI: 10.3390/iot4020005
- [263] H. Yar, A. S. Imran, Z. A. Khan, M. Sajjad, and Z. Kastrati, “Towards smart home automation using iot-enabled edge-computing paradigm,” *Sensors*, vol. 21, no. 14, p. 4932, 2021. DOI: 10.3390/s21144932
- [264] M. Domb, “Smart home systems based on internet of things,” *Internet of Things (IoT) for Automated and Smart Applications*, vol. 25, pp. 1–13, 2019. DOI: 10.5772/intechopen.84894

- [265] F. Cicirelli, G. Fortino, A. Giordano, A. Guerrieri, G. Spezzano, and A. Vinci, “On the design of smart homes: A framework for activity recognition in home environment,” *Journal of Medical Systems*, vol. 40, no. 9, p. 200, 2016. DOI: 10.1007/s10916-016-0549-7
- [266] P. Lyle, H. Korsgaard, and S. Bødker, “What’s in an ecology? a review of artifact, communicative, device and information ecologies,” in *Proceedings of the 11Th Nordic Conference on Human-Computer Interaction: Shaping Experiences, Shaping Society*, 2020, pp. 1–14. DOI: 10.1145/3419249.3420185
- [267] B. J. McNely, “Exploring a sustainable and public information ecology,” in *Proceedings of the 28Th ACM International Conference on Design of Communication*, 2010, pp. 103–108. DOI: 10.1145/1878450.1878468
- [268] H. Stopps, B. Huchuk, M. F. Touchie, and W. O’Brien, “Is anyone home? a critical review of occupant-centric smart hvac controls implementations in residential buildings,” *Building and Environment*, vol. 187, p. 107369, 2021. DOI: 10.1016/j.buildenv.2020.107369
- [269] D. Ferraiolo, J. Cugini, and D. R. Kuhn, “Role-based access control (rbac): Features and motivations,” in *Proceedings of 11Th Annual Computer Security Application Conference*, 1995, pp. 241–48.
- [270] M. Windl, P. Thalhammer, D. Müller, A. Schmidt, and S. S. Feger, “Privacyhub: A functional tangible and digital ecosystem for interoperable smart home privacy awareness and control,” in *Proceedings of the 2025 CHI Conference on Human Factors in Computing Systems*, 2025, pp. 1–15. DOI: 10.1145/3706598.3713517
- [271] N. Abdi, X. Zhan, K. M. Ramokapane, and J. Such, “Privacy norms for smart home personal assistants,” in *Proceedings of the 2021 CHI Conference on Human Factors in Computing Systems*, 2021, pp. 1–14. DOI: 10.1145/3411764.3445122

- [272] L. De Russis and F. Corno, “Homerules: A tangible end-user programming interface for smart homes,” in *Proceedings of the 2015 ACM CHI Conference Extended Abstracts on Human Factors in Computing Systems*, 2015, pp. 2109–2114.
- [273] A. Berger, A. Kurze, A. Bischof, J. J. Benjamin, R. Y. Wong, and N. Merrill, “Accidentally evil: On questionable values in smart home co-design,” in *Proceedings of the 2023 CHI Conference on Human Factors in Computing Systems*, ser. CHI ’23, Hamburg, Germany: Association for Computing Machinery, 2023, ISBN: 9781450394215. DOI: 10.1145/3544548.3581504 [Online]. Available: <https://doi.org/10.1145/3544548.3581504>
- [274] K. K. A. Alghythee, A. Hrcic, K. Singh, S. Kunisetty, Y. Yao, and N. Soni, “Towards understanding family privacy and security literacy conversations at home: Design implications for privacy literacy interfaces,” in *Proceedings of the 2024 CHI Conference on Human Factors in Computing Systems*, ser. CHI ’24, Honolulu, HI, USA: Association for Computing Machinery, 2024, ISBN: 9798400703300. DOI: 10.1145/3613904.3641962 [Online]. Available: <https://doi.org/10.1145/3613904.3641962>
- [275] X. Xue, X. Li, B. Jia, J. Du, and X. Fu, “Who should hold control? rethinking empowerment in home automation among cohabitants through the lens of co-design,” in *Proceedings of the 2024 CHI Conference on Human Factors in Computing Systems*, ser. CHI ’24, Honolulu, HI, USA: Association for Computing Machinery, 2024, ISBN: 9798400703300. DOI: 10.1145/3613904.3642866 [Online]. Available: <https://doi.org/10.1145/3613904.3642866>
- [276] R. Menezes, J. Schuler, K. Zhu, O. Rogalski, and I. Gupta, “Knox: Fortifying smart spaces with safety guarantees,” in *45th International Symposium on Reliable Distributed Systems (SRDS)*, 2026.

- [277] S. B. Ahsan, R. Yang, S. A. Noghabi, and I. Gupta, “Home, SafeHome: Ensuring a safe and reliable home using the edge,” in *2nd USENIX Workshop on Hot Topics in Edge Computing (HotEdge 19)*, 2019.
- [278] E. King, H. Yu, S. Lee, and C. Julien, “Sasha: Creative goal-oriented reasoning in smart homes with large language models,” *Proceedings of the ACM on Interactive, Mobile, Wearable and Ubiquitous Technologies*, vol. 8, no. 1, pp. 1–38, 2024.
- [279] D. Abrokwa, S. Das, O. Akgul, and M. L. Mazurek, “Comparing security and privacy attitudes among u.s. users of different smartphone and smart-speaker platforms,” in *Seventeenth Symposium on Usable Privacy and Security (SOUPS 2021)*, 2021, pp. 139–158.
- [280] J. C. Hou, Q. Wang, B. K. AlShebli, L. Ball, S. Birge, M. Caccamo, C.-F. Cheah, E. Gilbert, C. A. Gunter, E. L. Gunter, C.-G. Lee, K. Karahalios, M.-Y. Nam, N. Narasimhan, R. Chaudhri, L. Sha, W. Shin, S. Yu, Y. Yu, and Z. Zeng, “PAS: A wireless-enabled, sensor-integrated personal assistance system for independent and assisted living,” in *Joint Workshop on High Confidence Medical Devices, Software, and Systems and Medical Device Plug-and-Play Interoperability (HCMDSS/MDPnP)*, IEEE, 2007, pp. 64–75. DOI: 10.1109/HCMDSS-MDPnP.2007.13
- [281] B. K. AlShebli, E. Gilbert, and K. Karahalios, “The social implications of an assisted living reminder system,” in *Online Communities and Social Computing (OCSC), Held as Part of HCI International*, ser. Lecture Notes in Computer Science, vol. 4564, Springer, 2007, pp. 239–249. DOI: 10.1007/978-3-540-73257-0_27
- [282] B. Tao, M. Masood, I. Gupta, and D. Vasisht, “Transmitting, fast and slow: Scheduling satellite traffic through space and time,” in *29th Annual International Conference on Mobile Computing and Networking (MobiCom)*, 2023. DOI: 10.1145/3570361.3592521

- [283] B. Tao, O. Chabra, I. Janveja, I. Gupta, and D. Vasisht, “Known knowns and unknowns: Near-realtime earth observation via query bifurcation in serval,” in *21st USENIX Symposium on Networked Systems Design and Implementation (NSDI 24)*, 2024, pp. 809–824. [Online]. Available: <https://www.usenix.org/conference/nsdi24/presentation/tao>
- [284] M. Masood, S. Kannan, O. Chabra, D. Vasisht, and I. Gupta, *SILC: Lookahead caching for short-form video delivery systems*, 2026. arXiv: 2605.05188 [cs.NI]. [Online]. Available: <https://arxiv.org/abs/2605.05188>
- [285] L. Tang, C. Bhandari, Y. Zhang, A. Karanika, S. Ji, I. Gupta, and T. Xu, “Fail through the cracks: Cross-system interaction failures in modern cloud systems,” in *Proceedings of the Eighteenth European Conference on Computer Systems*, ser. EuroSys ’23, New York, NY, USA: Association for Computing Machinery, 2023, pp. 433–451, ISBN: 9781450394871. DOI: 10.1145/3552326.3587448

Appendix A: Control in Context—Supplemental Material

A.1 Main Survey Questions

A.1.1 Consent Block

Q1 You are being asked to participate in a voluntary research study. The purpose of this study is to build a taxonomy of smart home user workloads and failures for both centralized (using a smart home hub) and per-device control schemes. Your responses will provide us with information regarding existing approaches to controlling smart home device infrastructures, and the reasons for opting to use certain control schemes.

Participating in this study will involve answering a set of survey questions about your experiences and current smart home setups. Participation will last approximately 20 minutes. We will not be asking you to use any of these devices in front of us, but instead we will be asking you to recount experiences with these devices. There are no foreseeable risks to you beyond those normally encountered in your daily life while participating in this survey. If you agree to participate, you will be asked to check "I consent" at the bottom of this page indicating that you have read the following form and have been shown the goals of this study.

Principal Investigator Name and Title: Removed for anonymity

What procedures are involved? The study procedures are an online survey that you will be asked to complete. In this survey, you will be asked about your smart home setup, including the kinds of devices you own and use, as well as how you control them. We will then ask some questions about specific experiences with these devices, and how these experiences have affected your perception of these devices and corresponding control schemes.

This research will be performed online, using an online survey software (Qualtrics). Participation will consist of one single survey, that will take approximately 20 minutes to

complete.

Will my study-related information be kept confidential? We will use all reasonable efforts to keep your personal information confidential, but we cannot guarantee absolute confidentiality. When this research is discussed or published, no one will know that you were in the study. But, when required by law, identifying information may be seen or copied by: a) The Institutional Review Board that approves research studies; b) The Office for Protection of Research Subjects and other university departments that oversee human subjects research; c) University and state auditors responsible for oversight of research.

Will I be reimbursed for any expenses or paid for my participation in this research? You will be compensated 4\$ USD upon completion of this survey.

Can I withdraw or be removed from the study? If you decide to participate, you are free to withdraw your consent and discontinue participation at any time. Your participation in this research is voluntary. Your decision whether to participate, or to withdraw after beginning participation, will not affect your current or future dealings with [Removed for Anonymity].

The researchers also have the right to stop your participation in this study without your consent if they believe it is in your best interests, you were to object to any future changes that may be made in the study plan.

Will data collected from me be used for any other research? Your de-identified information could be used for future research without additional informed consent.

Who should I contact if I have questions? Removed for anonymity

What are my rights as a research subject? Removed for anonymity

Please print this consent form if you would like to retain a copy for your records.

I have read and understand the above consent form. I certify that I am 18 years old or older. By clicking the “I consent” button to enter the survey, I indicate my willingness to voluntarily take part in this study.

Answer type: Multiple choice. Options were: “I consent.” and “I do not consent.” If a

participant chose “I do not consent.” they were automatically redirected to the end of the survey.

A.1.2 Screening Block

Q2 How many Smart Devices do you currently own? As a reminder, Smart Devices are defined as: Everyday objects made intelligent with network capabilities to execute tasks either automatically or on demand. Some examples of these devices are, but not limited to: smart plugs/switches, smart lights, smart door locks, smart thermostats, smart vacuums, window/door sensors, motion sensors, smart smoke alarms, smart cameras/security systems, smart TVs, smart speakers (amazon echo, google home), etc. Excludes smartphones, tablets, computers.

Answer type: Multiple choice. Options were: “I do not own any smart devices”, “1”, “2”, “3”, “4”, “5 or more smart devices”.

A.1.3 Intro Block

Thank you for participating in this survey. The purpose of this survey is to understand user experiences and perceptions with smart home device control, and how these perceptions change over time. Your responses will provide valuable information to help answer our research questions. Please keep in mind that this survey is not a test of your knowledge, but a way to learn about your experiences; there are no right or wrong answers.

For your reference, here are some definitions to terms you may encounter in this survey:

Smart Devices: Everyday objects made intelligent with network capabilities to execute tasks either automatically or on demand. Some examples of these devices are, but not limited to: smart plugs/switches, smart lights, smart door locks, smart thermostats, smart vacuums, window/door sensors, motion sensors, smart smoke alarms, smart cameras/security systems, smart TVs, smart speakers (amazon echo, google home), etc. Excludes smartphones, tablets, computers.

Smart Device Control: The ability to view and change the current state of a smart device. The smart device itself, a physical controller or an application may be access points to control the smart device. Typically, smart devices are paired with the control medium, if that is different from the device itself, before the user can control the device. For example, you can control a smart lock on the front door either directly or with your phone.

For the purposes of this study, we will differentiate between different ways you may control your smart home devices:

Centralized Control: Control of multiple smart devices from a central service (e.g., digital twin) or device (e.g., smart hub), that has a global view of all devices' states and access to modify the state of any and all of the devices. All smart devices must initially be paired with the central service by the user. Some examples of centralized controllers are: Amazon Echo, Google Home mini, Apple HomePod for devices, and Amazon Alexa, Google Home, Google Assistant and Apple HomeKit for applications.

Per-device control: Control of smart devices by pushing or rotating buttons on an individual device itself or by using an application that the device vendor has provided the customers. The user must pair the smart device with the device vendor if they desire controlling it by the provided app. Some examples of per-device controllers include but are not limited to: Philips Hue Bridge, Wink+GE In-Wall Smart Switch for devices, Philips Hue, ecobee for applications.

Routine: An automation that involves smart devices. A set of actions, and potentially a set of triggering conditions, are specified. Each action changes the current state of a smart device. Each triggering condition checks the current state of a smart device or the current date and time; triggering conditions may be combined with a conjunction or a disjunction. A routine may also be manually triggered by a user, rather than triggering clauses. Other names for a routine in current products are: recipe, scene, schedule.

Lastly, we define the following term(s):

Troubleshooting: Tracing and correcting faults in a mechanical or electronic system.

An example of a troubleshooting process is the following:

1. You are at home and your smart vacuum no longer responds to your control through your Amazon Echo device.
2. You check that your smart vacuum base is connected to a power outlet; it is.
3. You check that your smart vacuum is connected to the WiFi in your home; it is.
4. You check that your smart phone has access to the internet when connected through the home WiFi; it does not.
5. You check that your home router has access to the internet; it does not.
6. You reboot your home router and wait for 5 minutes.
7. Your smart vacuum responds to your commands through your Amazon Echo device.

Please remember that you are answering on behalf of your household.

Q3 Please enter your prolific ID.

Answer type: Text entry (single line).

A.1.4 General Smart Home Setup Questions

Q4 Which of the following smart home devices do you have powered on where you are living now? In the first column, select all that apply. For each owned device, in the second column, write how many you own and in the third column check each location in the home where you have the device. Finally, in the fourth column, describe the device's primary use in your home.

Recall that smart devices are defined as: "Everyday objects made intelligent with network capabilities to execute tasks either automatically or on demand." If you are at home, feel free to walk around and look at devices to jog your memory.

Answer type: Table filling. Each row represented a smart device kind out of plug/switch, light, lock, thermostat, vacuum/mop, window/door sensor, motion sensor, smoke alarm, camera, security system, TV/casting device, speaker, tag, health device, washing machine/ dryer, dishwasher, fridge, coffee maker, oven/range, trash can, other. There was a row for "I do not own any smart devices" as well. Each device kind was accompanied by a couple of representative devices. The first column had checkmarks to be clicked if a device kind was owned. The second column asked for the number of owned devices for each device kind. Next, we asked in a separate group of columns for participants to pick the spaces their devices of each kind were located at in their home; options were: entrance hall, living room, dining room, kitchen, master bedroom, children's bedroom, guestroom, study, wardrobe, bathroom, basement, laundry room, hallway, entertainment/game room, attic, garage, porch (front/back), yard/garden/pool, driveway, and other. The final column asked participants for the primary use of the device kind.

Q5 If you selected other, for each device: Please list the number of devices, their type, their location in your home, and their main use.

Answer type: Text entry (multiple lines).

Q6 Look at your phone. How many smart home device related applications do you have installed on your phone?

Answer type: Multiple choice. Options were: "0", "1-4", "5-8", "More than 8".

If Q6 answer was not "0": Q7 Please list the smart home device related applications on your phone.

Answer type: Text entry (essay text box).

Q8 When was the last time you purchased a smart device for your home?

Answer type: Multiple choice. Options were: "Less than one month ago", "Less than

six months ago”, “Less than one year ago”, and “More than one year ago”.

Q9 What was one of the last smart devices you purchased for your home? Why did you purchase it?

Answer type: Text entry (essay text box).

Q10 To what extent do you agree with this statement about the device you mentioned in the previous question: “I was able to accomplish the goals I had in mind when I first purchased this device”?

Answer type: Multiple choice. Options were: “Strongly disagree”, “Somewhat disagree”, “Neither agree nor disagree”, “Somewhat agree”, and “Strongly agree”.

Q11 For any goal that you were not able to accomplish, please list the reasons.

Answer type: Text entry (essay text box).

Q12 For any goal that you were able to accomplish, please tell us how you reached them.

Answer type: Text entry (essay text box).

A.1.5 Attention Check (1)

Q13 Please answer the following question: What is $7+4$?

Answer type: Multiple choice. Options were: “15”, “10”, “11”, and “19”.

A.1.6 Device Control Questions

Q14 Pick a specific device in your home, and write down that device below.

Answer type: Text entry (single line).

The next few questions deal with this device that you mentioned above.

Q15 How often is the device you listed above in use (manually or via automation)?

Answer type: Multiple choice. Options were: “At least three times a week”, “At least once a week”, “At least once a month”, and “Only a couple times since I got it”.

Q16 What is the way the device you listed above is most commonly used?

Answer type: Multiple choice (multiple answers allowed). Options were: “Routines (defined as ‘An automation that involves smart devices. A set of actions, and potentially a set of triggering conditions, are specified. Each action changes the current state of a smart device’, “Central Control (defined as ‘Control of smart devices from a central service (e.g., digital twin) or device (e.g., smart hub), that has a global view of all devices’ states and access to modify the state of any and all of the devices.’)”, “Per-Device Control (defined as ‘Control of smart devices by pushing or rotating buttons on an individual device itself or by using an application that the device vendor has provided the customers.’)”, “Voice Command” and “Content streaming casting”.

Q17 Why do you choose to control the device you listed above this way(s)?

Answer type: Text entry (essay text box).

Q18 Is there any device in your home that can be controlled in multiple ways? (this can be the same device that you talked about on the previous page)

Answer type: Multiple choice. Options were: “No”, “Yes”, and “I don’t remember”.

If Q18 answer was “Yes”: Q19 If there are multiple devices that can be controlled in multiple ways, think about one of these devices. What device is that?

Answer type: Text entry (essay text box).

If Q18 answer was “Yes”: Q20 How do you decide between the different options of controlling this device?

Answer type: Text entry (essay text box).

Q21 Are you aware that central smart home managers exist? Examples include Amazon Alexa, Google Home, Apple Home.

Answer type: Multiple choice. Options were: “Yes” and “No”.

If Q21 answer was “Yes”: Q22 How did you learn about central smart home managers?

Answer type: Multiple choice (multiple answers allowed). Options were: “Internet”, “Social media”, “Streaming services”, “Physical/Online stores”, “Word of mouth”, and “Other (please specify)” with a text box next to it.

A.1.7 Device Exceptions (Blame Placement) Questions

Q23 Have you had any experience(s) when one or more of your smart devices did not work as planned?

Answer type: Multiple choice. Options were: “No”, “Yes”, and “I don’t remember”.

If Q23 answer was “Yes”: Q24 Can you describe an experience from a specific time when things didn’t go as planned?

Answer type: Text entry (essay text box).

If Q23 answer was “Yes”: Q25 Which of the following statements represents your current experiences with smart devices? *Answer type:* Multiple choice. Options where: “I am having the same kinds of issues again and again”, “I encounter a specific kind of issue just once and the next ones are unrelated”, “I don’t remember if there are patterns among the issues I come across”, and “I am not encountering any issues with my smart devices”.

If Q23 answer was “Yes” and Q25 answer was not “I am not encountering any issues with my smart devices”: Q26 What was your reaction to any smart device failures when you started noticing them? Can you describe the reasons that caused this reaction?

Answer type: Text entry (essay text box).

Q27 Have you ever returned or stopped using a smart device because it didn't work the way you planned?

Answer type: Multiple choice. Options were: "No", "Yes", and "I don't remember".

If Q27 answer was "Yes": Q28 What smart devices did you ever return or stop using?

Answer type: Text entry (essay text box).

A.1.8 Troubleshooting Questions

If Q23 answer was "Yes": Q29 How did you respond when your device did not work as expected?

Answer type: Multiple choice (multiple answers allowed). Options were: "I tried to figure out what went wrong", "I tried to fix the issue/prevent it from happening again", "I stopped using the device", "I gave up on that functionality", "I don't remember", and "Not applicable".

If Q29 answer "I tried to fix the issue/prevent it from happening again" was selected: Q30 Describe your troubleshooting process as a sequence of steps. Recall that troubleshooting is defined as: "The process of tracing and correcting faults in a mechanical or electronic system."

Answer type: Text entry (essay text box).

If Q23 answer was "Yes": Q31 Have you ever given up on using a device in your home?

Answer type: Multiple choice. Options were: "No", "Yes", and "I don't remember".

If Q31 answer was "Yes": Q32 How did you reach that point?

Answer type: Text entry (essay text box).

A.1.9 Attention Check (2)

Q33 Please answer the following question: What is 15-11?

Answer type: Multiple choice. Options were: “10”, “4”, “8”, and “6”.

A.1.10 Questions about Other Spaces

Q34 Are there any more homes or spaces with smart devices that you are the main maintainer of? If so, how many?

Answer type: Multiple choice. Options were: “No”, “Yes, 1 other space”, and “Yes, 2 or more other spaces”.

If Q34 answer was not “No”: Q35 Can you describe where devices are placed in these spaces? Can you describe where devices are placed in these spaces?

Answer type: Text entry (multiple lines).

If Q34 answer was not “No”: Q36 Do you think there are any similarities between the different spaces’ setups?

Answer type: Multiple choice. Options were: “No”, “Yes”, and “I don’t remember”.

If Q34 answer was not “No” and Q36 answer is not “I don’t remember”: Q37

Please explain your answer to the previous question.

Answer type: Text entry (multiple lines).

A.1.11 Demographic Block

Q38 What is your gender?

Answer type: Multiple choice. Options were: “Male”, “Female”, “Non-binary/third gender”, “Prefer not to say”, and “Prefer to self-describe” with a text box next to it.

Q39 How would you describe your current living situation?

Answer type: Multiple choice. Options were: “House”, “Apartment”, “Condominium”, “Town Home”, “Prefer not to say”, and “Other (please specify)” with a text box next to it.

Q40 What is the highest degree or level of school that you have completed?

Answer type: Multiple choice. Options were: “Some high school, no diploma”, “High school graduate, diploma or equivalent (for example: GED)”, “Some college credit, no degree”, “Associate’s degree”, “Bachelor’s degree”, “Post-graduate degree”, “Prefer not to say”, and “Other (please specify)” with a text box next to it.

Q41 What is your race?

Answer type: Multiple choice. Options were: “White”, “Black or African American”, “American Indian or Alaska Native”, “Asian”, “Native Hawaiian or Pacific Islander”, and “Hispanic or Latino”.

Q42 What is your age?

Answer type: Multiple choice. Options were: “less than 18 years old”, “18-24”, “25-34”, “35-44”, “45-54”, “55-64”, and “65+”.

A.1.12 Interview Interest Block

Q43 Would you be interested in doing a follow-up interview regarding your smart home device and control experience, for additional compensation? If you indicate you are interested, we will reach out via your prolific ID.

Answer type: Multiple choice. Options were: “Yes” and “No”.

A.1.13 End of Survey

We thank you for your time spent taking this survey. Your response has been recorded.

A.2 Interview Script

Thank you so much for agreeing to participate in our interview. The purpose of this interview is to understand user experiences and perceptions with smart home device control, and how these perceptions change over time. Your responses will provide valuable information to help answer our research questions. Please keep in mind that this interview is not a test of your knowledge, but a way to learn from your experiences; there are no right or wrong answers. Some of these questions are meant to build upon your responses in the survey you previously completed. However, feel free to correct information provided in the survey or to add more context.

I hope you had time to review a copy of the consent form for this study previously, before replying that you are willing to participate, but we want to be sure that you understand everything – any questions about the consent form?

Answer any questions regarding the consent form that they may have...

My collaborator [INSERT NAME] is here with me today and will be taking some notes during the interview.

As specified in the consent form, we will be audio recording and screen capturing this interview. After we get your permission, we will use Zoom to record the video and audio of this call. We will not be storing your name and personal information without explicitly telling you ahead of time. After today's interview, we will replace your personal information with a user code and keep your identity confidential. We will be referring to this user code when discussing and analyzing the data. You will not be identified in any publications.

If at any point you can't hear me, or have connection issues, feel free to use the text chat feature to let me know. If you get disconnected from the video chat, please rejoin the meeting when your connection stabilizes (we'll wait for you).

I will begin recording now, is that okay with you?

A.2.1 Smart Home Setup

Talk about the specific device that the participant mentioned in the survey: In the survey, you mentioned that you control [INSERT DEVICE HERE] primarily through [INSERT CONTROL SCHEME HERE]. Have you ever tried controlling this device differently?

- If yes: How did that experience go?
- If no: Do you feel that this way is sufficient for all your current goals with this device?
 - If not: Can you think of an experience where you felt like how you control this device has led to problems?
 - If yes: Do you see a situation where your goals may change with this device, and this way of controlling it is no longer sufficient?

Do you think that there is an ideal way to control your devices that may or may not exist currently?

- If yes: can you describe it?
- If no: do you think that central control could be ideal? Why or why not?
 - If yes: What device is your voice command set up with? What do you dislike most about your voice commands? How do you think about the relationship between your central manager and the device-specific app?

Devices in a smart home can be set up in a way that they work together. For example, when the AC starts working, doors and windows are closed. Do those capabilities sound familiar? Have you set up your smart home in a way that your devices work together?

- If not: Do you feel like you are able to accomplish your goals without this? How could you benefit from a connected setup where devices could work in coordination with each other? How might that kind of setup concern you, if at all?

- If yes: How do you benefit from such a setup? Do you feel like this option is a gamechanger in any way?
- How might a setup that includes sensors benefit you?
- Did you set up your routines on your own?
- Is there any option to set it up on the device app itself?
- Why did you pick the central manager to set it up on?

Have you ever encountered devices that wouldn't effectively integrate into your existing smart home? By effectively, I mean that device collaborations are set up and proceed seamlessly.

- If yes: What did you do?
- Do you ever consider how a device you might buy will work with other devices already in your home before purchasing it?
- Do you think about how you will control a device before you purchase it? Did you have specific goals in mind before you purchased a device?
 - If yes: Have you ever experienced a situation where your expectation of controlling a device was different than reality?
 - * If yes: Has this caused you to change your expectations or goals with that specific device?
 - If yes: Did your goals change after you purchased your device?

A.2.2 Smart Home Device Control

Talk about the specific device that the participant mentioned in the survey:

- In the survey, you mentioned that you control [INSERT DEVICE HERE] primarily through [INSERT CONTROL SCHEME HERE]. Have you ever tried controlling this device differently?
 - *If yes*: How did that experience go?
 - *If no*: Do you feel that this way is sufficient for all your current goals with this device?
 - * *If not*: Can you think of an experience where you felt like how you control this device has led to problems?
 - * *If yes*: Do you see a situation where your goals may change with this device, and this way of controlling it is no longer sufficient?
- Do you think that there is an ideal way to control your devices that may or may not exist currently?
 - *If yes*: can you describe it?
 - *If no*: do you think that central control could be ideal? Why or why not?
 - * *If yes*: What device is your voice command set up with? What do you dislike most about your voice commands? How do you think about the relationship between your central manager and the device-specific app?
- Devices in a smart home can be set up in a way that they work together. For example, when the AC starts working, doors and windows are closed. Do those capabilities sound familiar?
- Have you set up your smart home in a way that your devices work together?
 - *If not*: Do you feel like you are able to accomplish your goals without this? How could you benefit from a connected setup where devices could work in coordination with each other? How might that kind of setup concern you, if at all?

- *If yes:* How do you benefit from such a setup? Do you feel like this option is a gamechanger in any way?
- How might a setup that includes sensors benefit you?
- Did you set up your routines on your own?
- Is there any option to set it up on the device app itself?
- Why did you pick the central manager to set it up on?
- Have you ever encountered devices that wouldn't effectively integrate into your existing smart home? By effectively, I mean that device collaborations are set up and proceed seamlessly.
 - *If yes:* What did you do?
 - Do you ever consider how a device you might buy will work with other devices already in your home before purchasing it?
 - Do you think about how you will control a device before you purchase it? Did you have specific goals in mind before you purchased a device?
 - * *If yes:* Have you ever experienced a situation where your expectation of controlling a device was different than reality?
 - *If yes:* Has this caused you to change your expectations or goals with that specific device?
 - * *If yes:* Did your goals change after you purchased your device?
- *If they have more than 8 applications:* You mentioned that you have a lot of applications on your phone for controlling smart devices. How do you manage this?

A.2.3 Smart Home Errors and Exceptions

If the participant stated that they're experiencing problems with their smart devices

- You mentioned that you experience issues with smart devices.
- How do you verify that an action actually takes place on a smart device?
- Can you tell me about the most memorable or recent failure of a smart device? How did it happen the first time? Do you see a pattern in the failures?
- Has your trust in any specific devices been affected after experiencing issues with them?
- Have any issues affected your willingness to purchase specific types of devices?

– *If yes:* Which ones? What was the issue you experienced with these devices?

If the participant mentioned that they have encountered problems with (a) specific pair(s) of devices from different vendors:

- In the survey, you told us that some vendors' devices do not work well with each other. Specifically, you experienced problems while trying to integrate/integrating [INSERT DEVICE A] with [INSERT DEVICE B]. Can you share more instances of the same problem with different pairs of devices? What were you trying to accomplish with each pairing?

If not:

- Have you encountered any issues in making devices from different vendors work together? What were those issues?

– *If yes:* What were those issues?

If the participant mentioned that they have stopped using or returned a specific device because it wasn't working for them:

- You mentioned that you stopped using or returned [insert specific device] because it wasn't working properly. Did you replace that device in your home with something else?

- *If yes*, what did you consider when choosing a replacement device?
- *If not*, did you decide to not try and accomplish what you wanted to with [insert specific device]?

A.2.4 Smart Home Troubleshooting

- In the survey, you talked about your troubleshooting process as this set of steps
[Insert answer from the survey]
 - *Go through the steps, and if any of them seem particularly interesting or not detailed enough, ask:* Could you explain a little more about what you mean when you say you [DO X]?
 - Do you find that there is an approach that universally helps you fix problems that come up in your smart home?
 - What was the most difficult and memorable process you went through to get a device or automation to work in your smart home?

If the participant mentions that they have given up on a device before:

- You mentioned that you gave up on getting [insert device name] to work in your home. Have you decided to get other devices to work on your home?
 - Was the process of getting to this point similar to what you described for [insert device name]?

Appendix B: Other Work

In this appendix I briefly describe other research projects that I conducted during my Ph.D. in parallel with this thesis.

Cross-System Interaction Failures [285]. Modern cloud deployments are orchestrations of independent systems—data processing engines, storage layers, resource managers—each specializing in one service. Their reliability therefore depends not only on each system in isolation but on the interplay between them, and a growing share of production incidents manifests at these boundaries rather than inside any single component. We termed this failure mode *cross-system interaction* (CSI) failures and characterized it by analyzing eleven CSI-failure-induced incidents at Google, Azure, and AWS alongside 120 CSI failure cases drawn from seven widely co-deployed open-source systems. The study identifies *discrepancies* between interacting systems as the underlying root cause—failures that cannot be understood by examining any one system alone—and argues for cross-system testing and verification. Cross-testing the Spark–Hive data plane exposed fifteen previously unknown discrepancies.