FLUID-IoT : Flexible and Granular Access Control in Shared IoT Environments via-UI-Level Control Distribution

Sunjae Lee[*]
sunjae1294@kaist.ac.kr
School of Computing, KAIST
Daejeon, Republic of Korea

Minwoo Jeong[*]
hayan@kaist.ac.kr
Graduate School of Information Security, KAIST
Daejeon, Republic of Korea

Daye Song
sdy992000@kaist.ac.kr
School of Computing, KAIST
Daejeon, Republic of Korea

Junyoung Choi
joonchoi518@kaist.ac.kr
School of Computing, KAIST
Daejeon, Republic of Korea

Seoyun Son
seoyun.son@kaist.ac.kr
School of Computing, KAIST
Daejeon, Republic of Korea

Jean Y. Song[†]
jeansong@dgist.ac.kr
Electrical Engineering and Computer Science, DGIST
Daegu, Republic of Korea

Insik Shin[†]
ishin@kaist.ac.kr
School of Computing, KAIST
Daejeon, Republic of Korea

ABSTRACT

The rapid growth of the Internet of Things (IoT) in shared spaces has led to an increasing demand for sharing IoT devices among multiple users. Yet, existing IoT platforms often fall short by offering an all-or-nothing approach to access control, not only posing security risks but also inhibiting the growth of the shared IoT ecosystem. This paper introduces FLUID-IoT, a framework that enables flexible and granular multi-user access control, even down to the User Interface (UI) component level. Leveraging a multi-user UI distribution technique, FLUID-IoT transforms existing IoT apps into centralized hubs that selectively distribute UI components to users based on their permission levels. Our performance evaluation, encompassing coverage, latency, and memory consumption, affirm that FLUID-IoT can be seamlessly integrated with existing IoT platforms and offers adequate performance for daily IoT scenarios. An in-lab user study further supports that the framework is intuitive and user-friendly, requiring minimal training for efficient utilization.

CCS CONCEPTS

• **Security and privacy** → *Domain-specific security and privacy architectures*; • **Human-centered computing** → **User interface management systems**.

[*]Both authors contributed equally to this research.
[†]Co-corresponding authors: Jean Y. Song, Insik Shin.

KEYWORDS

IoT-Security, Access Control, Multi-User IoT, UI distribution

ACM Reference Format:
Sunjae Lee, Minwoo Jeong, Daye Song, Junyoung Choi, Seoyun Son, Jean Y. Song, and Insik Shin. 2024. FLUID-IoT : Flexible and Granular Access Control in Shared IoT Environments via-UI-Level Control Distribution. In *Proceedings of the CHI Conference on Human Factors in Computing Systems (CHI '24), May 11–16, 2024, Honolulu, HI, USA.* ACM, New York, NY, USA, 16 pages. https://doi.org/10.1145/3613904.3641991

1 INTRODUCTION

The Internet of Things (IoT) has experienced rapid growth in recent years, with numerous devices being integrated into homes [5], workplaces [26], and various other multi-user domains [48]. As IoT devices become increasingly prevalent in various multi-user settings, sharing control of these devices with others has become a common occurrence [20, 46]. Accordingly, IoT platforms such as `Samsung SmartThings`, `Google Home`, and `Amazon Alexa` have developed features that enable users to invite others to access and manage their IoT devices within a shared environment [21, 35, 38].

However, while such features facilitate the easy sharing of IoT control, they lack sophisticated access control measures such as allowing guests to control only the light bulbs in their room or restricting access to the smart camera in the host's bedroom—raising privacy concerns [30] and making individuals hesitant to share their IoT devices with others [24, 46]. For example, an invitation to a Google Home setup grants guests control over all connected devices, even allowing them to view logs and change settings.

Prior work has attempted to enhance IoT access control by altering existing applications [40, 41] or creating new systems altogether [7, 51]. However, as these approaches require significant modifications to the conventional IoT stacks, most of them are limited to proposing conceptual interfaces or simulation results. Furthermore, previous studies generally concentrate on regulating

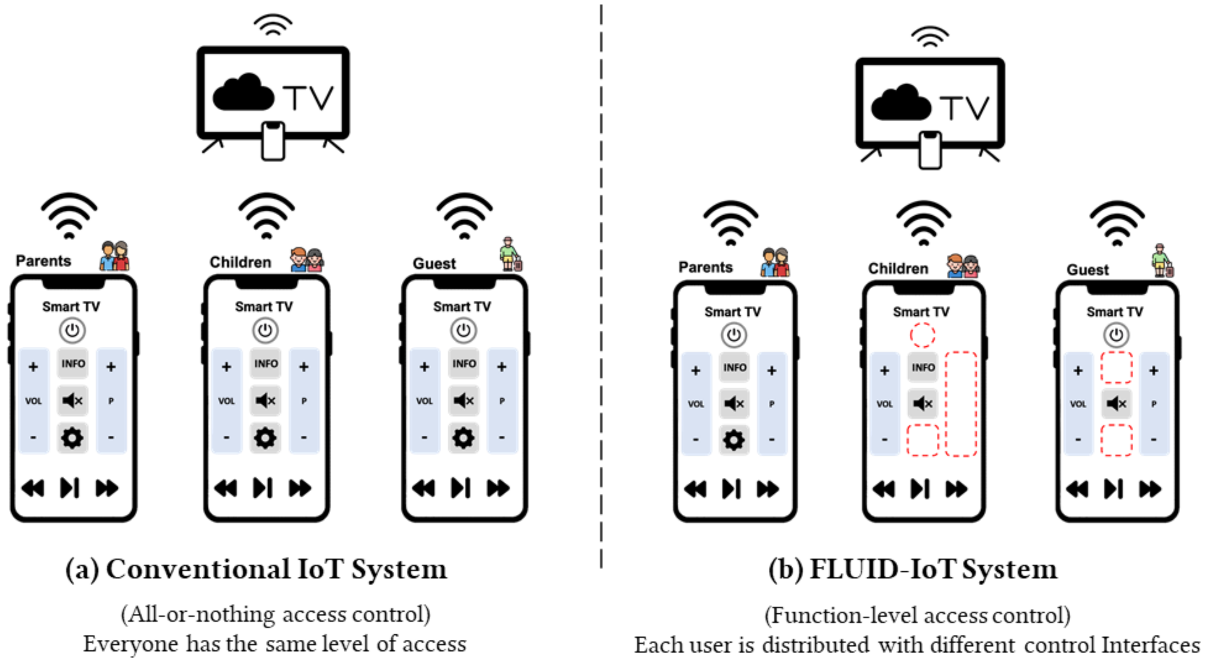

TV
Parents
Children
Guest
Smart TV
INFO
VOL
P
(a) Conventional IoT System
(All-or-nothing access control)
Everyone has the same level of access
(b) FLUID-IoT System
(Function-level access control)
Each user is distributed with different control Interfaces

Figure 1: Comparison between the conventional IoT system and FLUID-IoT system: (a) The conventional IoT system provides All-or-nothing access control where all three users (parents, children, guest) see the same control interfaces. (b) In contrast, FLUID-IoT provides fine-grained access control over individual functionalities of a device where each user can be allocated with different set of control interfaces. FLUID-IoT's advantage is in its flexibility where various arbitrary access control mechanisms could be easily implemented on existing IoT Platforms.

access to the device as a whole, overlooking the need and potential for finer-grained access control at the level of individual functionalities within each IoT device (think of power control, temperature control, fan control, and mode control of an air conditioner).

In this paper, we introduce FLUID-IoT, a framework designed to seamlessly integrate flexible and fine-grained multi-user access control to existing IoT platforms. FLUID-IoT introduces a novel multi-user User Interface (UI) distribution technique that converts an existing IoT app into a centralized control UI[1] distributor that sits between IoT devices and users to manage user permission and distribute control UIs to users based on their access permissions (Figure 1(b)). Our high-level insight put into this approach is that since IoT devices are typically controlled through graphical user interfaces (GUIs) as an intermediary, by regulating access to these interfaces, we can effectively control access to the devices themselves—*if a user cannot access the control interface, they cannot control the device.*

The benefits of this approach ares two folds. First, it enables granular control over specific device functions. Since UI is the most granular medium for interacting with specific functionality of an app, FLUID-IoT can control device access at its *finest granularity* by distributing different sets of UIs to different users. Some users may be distributed with one control UI, while others may receive two or three, depending on their access permissions. For example, in the case of a smart TV, a child might only be granted volume control buttons to keep them from accessing inappropriate channel, whereas parents could have full sets of control UIs, including power button, channel switch buttons, and volume controls (Figure 1(b)).

Second, our framework simplifies the challenges of IoT access management by eliminating the need to modify the underlying logic of the device connection pipelines. Since FLUID-IoT is designed to share *control interfaces* (e.g., power buttons, mode switch buttons, or fan speed sliders) rather than actual device connections themselves, implementing access control becomes as simple as determining which UI gets distributed to which user device. This way, we can *flexibly* implement various access control mechanisms on top of existing IoT platforms with minimal modifications–without modifying their underlying behavior logic.

We demonstrate the effectiveness of FLUID-IoT by implementing a prototype using a custom Android OS (Android Open Source Project) and unmodified off-the-shelf `Google Home` mobile application. Our coverage evaluation shows that FLUID-IoT can support the majority of prominent IoT platforms, including `Samsung SmartThings, Amazon Alexa, and LG ThinQ`. Our performance evaluation proves that FLUID-IoT provides near-instant UI response time, and significantly reduces the synchronization latency between user devices. We define three representative access control mechanisms; *i)* differential access control, *ii)* time-based access control, and *iii)* supervisory access control, and conducted an in-lab usability study with 17 participants using `Google Home` retrofitted with

[1]Within the context of this paper, 'control UI' refers to User Interfaces (UI) that are specifically designed for controlling IoT devices. E.g., volume control buttons, power buttons, and channel switch buttons for a smart TV

FLUID-IoT. The results show that FLUID-IoT provides high usability throughout different domains of the shared IoT environment.

To the best of our knowledge, FLUID-IoT is the first to introduce the methodology for fine-grained access control over individual functionality of devices and to propose a practical solution that can retrofit existing IoT platforms with access control features. More specifically, this paper makes the following contributions:

- We explore the need and potential for function-level fine-grained access control in multi-user IoT environment.
- We present the design and implementation of FLUID-IoT, which uses UI distribution technique to retrofit existing IoT apps with flexible function-level IoT access control.
- We implement and evaluate three different types of access control mechanism for different multi-user scenarios.
- We report results from user studies that demonstrate the effectiveness of the proposed framework from multiple angles—coverage, performance, and usability.

2 BACKGROUND AND RELATED WORK

In this section, we highlight the challenge present in the current shared IoT environment, provide a review of previous works to address this issue, and discuss the UI distribution technique that this work was inspired by.

2.1 Access Control Mechanisms in Existing IoT Platforms

Prominent IoT platforms such as Google Home, Samsung SmartThings, and Amazon Alexa enable users to share their IoT devices with others. A significant limitation, however, is that their access control operates in an all-or-nothing manner, offering either complete control or no control at all. For example, Google Home and SmartThings allow users invited to the shared IoT environment to control all connected devices without any oversight (Figure 2(a) and Figure 2(b)). Although some vendor-specific IoT apps, like Kwikset Kevo Smart Lock (Figure 2(c)) and August Smart Lock (Figure 2(d)) provide more rigorous access control mechanisms tailored to their specific device type, the prevailing access control practices in the current IoT landscape fall short of reflecting the diverse and complex user demands in multi-user IoT settings.

2.2 Enhancing Access Control Policies

Growing concerns about sharing IoT with untrusted parties have raised a range of security-related issues, including privacy leaks [11, 31], unauthorized device control [10], and user conflicts [24, 31]. Recent studies on smart home users also reveal that users clearly express their concerns regarding the need for rigorous access control mechanisms, even in their trusted smart home environment [9, 20, 50].

In response to these concerns, numerous studies have proposed access control policies for multi-user IoT environments [13, 20, 23, 25, 39, 46, 51]. Most notably, He et al. [20] present a large-scale user study that reveals home IoT users desire different access control capabilities for various situations and settings. They propose access control policies that take into account stakeholder relationships, individual device capabilities, and various contexts, such as time, device location, and users' proximity to the device. Moreover, in a study on community-based access control involving sharing IoT with more distantly related stakeholders like neighbors and friends, Tabassum et al. [46] demonstrate that users expressed unmet needs for finer-grained control. In addition, the study conveys that time- and event-based access controls are the two most desired control mechanisms.

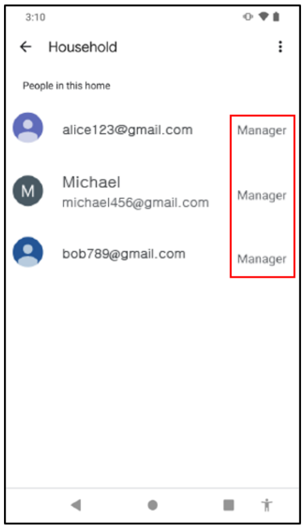
Given these in-depth studies on various access control policies, the goal of FLUID-IoT is to provide an access control framework that can easily integrate these policies into existing IoT platforms. A noteworthy caveat, however, is that prior studies tend to presume that the desire for access controls only occurs at the level of the device as a whole, either allowing or denying access to all of its functionalities at once. This device-level all-or-nothing approach is likely to pose similar risks as the conventional all-or-nothing control mechanisms when it comes to devices with diverse functionalities. In this regard, FLUID-IoT more specifically aims to enable proposed access control policies to operate at the granular level of individual device functionalities, verifying their needs and effectiveness along the way.

2.3 Multi-User Access Control Systems

Some researchers have proposed IoT systems that provide a predefined set of access control mechanisms. Zeng et al. [51] presented SmarterHome, a prototype IoT control app that uses Samsung SmartThings API to provide four different access control mechanisms. However, due to its reliance on the SmartThings API and SmartThings Cloud, it lacks the flexibility and granularity needed to address complex user demands beyond the SmartThings API boundaries. Sikder et al. [40, 41] proposed `Kratos` and `Kratos+` that employ a priority-based access-policy negotiation algorithm by modifying the open-source Samsung SmartThings App. Although they provide a wide range of access control policies, their system implementation requires intense modification of the existing IoT apps, and its applicability to other closed-source IoT platforms (e.g., Google Home, Amazon Alexa, LG ThinQ) remains under-explored.

In other works, researchers have proposed new IoT systems by modifying or replacing parts of IoT stacks. Boussard et al.[7] implemented `Future Spaces`, a system equipped with access control via Software-Defined LANs. Additionally, various studies [29, 45, 49] presented block-chain-based access control to ensure data security and prevent undesired access to IoT devices. However, these approaches often involve the development of new IoT platforms or require significant modifications to existing ones, which diminishes the practicality of the proposed solutions.

On the other hand, FLUID-IoT offers a practical and easily implementable solution for realizing various access control mechanisms on top of existing IoT platforms. Instead of building a new system from scratch [1, 7, 29, 37, 45, 49] or requiring existing ones to alter their internal behaviors [40, 41, 51], FLUID-IoT adds an extra access control layer on top of the established IoT stack, enabling off-the-shelf IoT apps to provide fine-grained access control over their connected devices.



3:10

Household

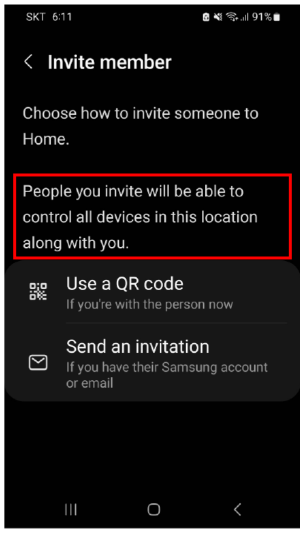
People in this home

alice123@gmail.com Manager

Michael
michael456@gmail.com Manager

bob789@gmail.com Manager

(a) Google Home with all-or-nothing access



SKT 6:11 91%

Invite member

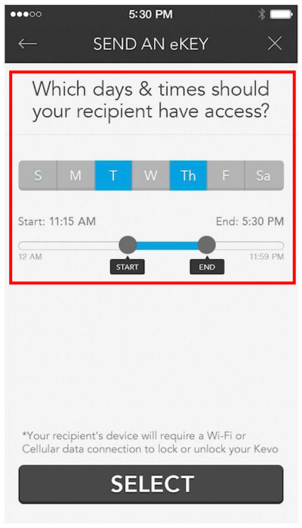
Choose how to invite someone to Home.

People you invite will be able to control all devices in this location along with you.

Use a QR code
If you're with the person now

Send an invitation
If you have their Samsung account or email

(b) Samsung SmartThings with all-or-nothing access



5:30 PM

SEND AN eKEY

Which days & times should your recipient have access?

S M T W Th F Sa

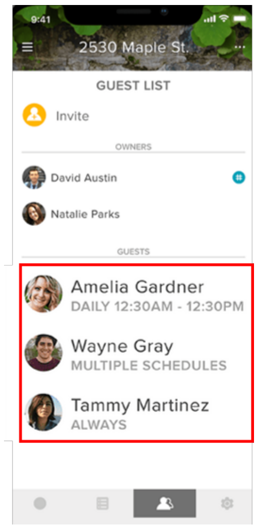
Start: 11:15 AM End: 5:30 PM

12 AM START END 11:59 PM

*Your recipient's device will require a Wi-Fi or Cellular data connection to lock or unlock your Kevo

SELECT

(c) Kiwikset Kevo Smart Lock with time-based access



9:41

2530 Maple St.

GUEST LIST

Invite

OWNERS

David Austin

Natalie Parks

GUESTS

Amelia Gardner
DAILY 12:30AM - 12:30PM

Wayne Gray
MULTIPLE SCHEDULES

Tammy Martinez
ALWAYS

(d) August Smart Lock with time-based access

Figure 2: Google Home (a) and Samsung SmartThings (b) offer all-or-nothing access control. While some vendor-specific IoT apps like Kiwikset (c) and August Home (d) offer more rigorous access control mechanisms tailored to their specific devices, they still fall short of reflecting the diverse and complex user demands in multi-user IoT settings.

2.4 UI Distribution Techniques

FLUID-IoT leverages UI distribution technique to selectively distribute IoT app's control interfaces to the users. UI distribution, also known as UI mirroring or UI casting, is a widely used technique that enables multiple devices or users to share control over a single app. Most renowned examples include Google Cast [18] and Apple AirPlay [4], which transmit multimedia content from a mobile device to other display devices. Other applications like Android Auto [16] and Apple CarPlay [3] allow apps on a user's mobile phone to display some of their interfaces on an automobile's infotainment display. However, their underlying UI distribution techniques require apps to modify their source code, which FLUID-IoT aims to avoid.

To support UI distribution across unmodified apps, several non-intrusive UI distribution frameworks have been introduced. *FLUID* [34] and FLUID-XP [28] presents a flexible and transparent UI distribution technique that allows off-the-shelf Android mobile apps to distribute their UI elements to another device. More distantly related works remix the original interfaces of applications to better support the diverse needs of users. Façades [44] and WinCuts [47] enables users to self-adjust desktop applications' interfaces by recombining existing graphical interfaces. A-Mash [27] lets users merge interfaces of multiple mobile apps to craft their own tailor-made all-in-one mobile app.

What sets FLUID-IoT's UI distribution apart from these approaches is its emphasis on multi-user interaction. Unlike previous frameworks that are geared toward single-user scenarios (i.e., one-to-one distribution settings or within-device UI remixing), FLUID-IoT enables one-to-many distribution, in which each user receives their own customized remix of UIs based on the distribution rules (i.e., access control policies). This significant enhancement necessitates an in-depth exploration of key challenges including synchronization, performance, and scalability, with the goal of delivering a responsive and consistent UI experience across multiple users.

3 FLUID-IOT METHODOLOGY

This section describes key design features of FLUID-IoT. FLUID-IoT is a multi-user UI distribution framework designed to retrofit existing IoT platforms with more granular and flexible permissions settings than those natively supported by their underlying platform.

3.1 Architecture Design

3.1.1 Concept. Figure 3 illustrates the system architecture of FLUID-IoT. FLUID-IoT consists of one host device (i.e., hub device) and multiple user devices. The host device is the only entity in the system that operates the IoT app and communicates with the IoT devices. Its chief role is to distribute the control UIs of the IoT app to the connected user devices and to communicate with IoT devices on behalf of all other user devices. Within the host device, FLUID-IoT framework integrates into existing IoT apps (e.g., Google Home). This requires either a minor modification to the IoT app's source code to include FLUID-IoT's application-level SDK or running a custom Android OS equipped with FLUID-IoT system-level framework. In contrast, the user devices require neither the IoT app nor the custom Android OS. Instead, they utilize the FLUID user application that receives and displays the control UIs from the host device. The user interaction with the UIs is done simply by forwarding the interaction event to the host device, where all the computation and communication actually take place.

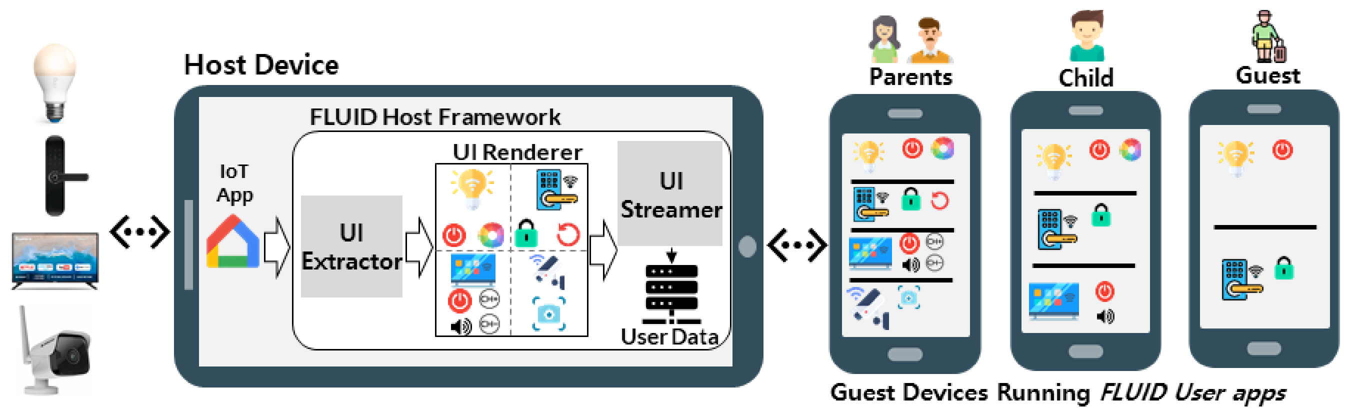


Figure 3: FLUID-IoT System architecture. In FLUID-IoT system, the host device is the only entity in the system that runs the IoT app and communicates with the IoT devices on behalf of the users. User devices run FLUID user app instead of the IoT app. Inside the host device, FLUID-IoT framework is integrated into existing IoT apps (e.g., Google Home) to act as a centralized UI distributor. FLUID-IoT framework extracts, renders, and streams each UI element of the IoT app to distribute UIs to FLUID user apps based on the users' permission level.

Overall, FLUID-IoT system mirrors the characteristics of an IoT hub architecture, where multiple IoT devices are connected to a single hub (i.e., a host device), and the hub communicates with the IoT devices on behalf of users. In this respect, FLUID-IoT is a framework that can seamlessly transform conventional IoT apps into a multi-user IoT hub. Such design choice has several advantages including *1)* ease of integration, *2)* ease of usage, and *3)* ease of synchronization.

3.1.2 Ease of integration. FLUID-IoT's host framework is designed to be easily integrated into conventional IoT apps. The framework consists of three main components: 1) *UI Extractor* that extracts the UIs from the IoT app, 2) *UI Renderer* that renders each extracted UI on an individual frame buffer, and 3) *UI Streamer* that streams rendered UIs to user devices. These components are designed to operate on layers (i.e., UI architecture and rendering pipeline) independent of the behavioral logic of mobile apps. Thus, they can be easily embedded into existing IoT applications without modifying the app's core internal logic.

3.1.3 Ease of usage. Over the years, IoT apps have evolved to support interaction with multiple devices through a single app [36]. FLUID-IoT takes this a step further by enabling users to interact with multiple IoT apps using a single FLUID user application. Since FLUID-IoT distributes the UIs in a form of the pixel, a universal, platform-independent data format, FLUID user application is compatible with any mobile IoT application. Users do not need to install multiple IoT apps on their devices; a single FLUID user app suffices for interacting with any IoT platform, as long as its native IoT app is running on the host device and has been properly integrated with FLUID-IoT. (More details in subsubsection 6.4.5).

3.1.4 Ease of synchronization. In conventional GUI-based IoT systems, each user device runs its own IoT app locally. When a user changes the state of an IoT device (e.g., turning on a bulb), the change isn't immediately reflected on other users' devices. Instead, the IoT device must first broadcast its updated state. The frequency of these broadcasts varies across platforms and devices, but this process inevitably causes a synchronization delay between user devices. For instance, toggling a light bulb on and off in systems like Google Home, Amazon Alexa, and Samsung SmartThings has been observed to result in synchronization delays of 3~5 seconds. Contrarily, FLUID-IoT eliminates this delay by running the IoT app on a host device. The visualized IoT state (i.e., UI) on all user devices are live projection of this app. Since there is no need for synchronization when there is only one entity (i.e., IoT app) that controls the shared resources (i.e., IoT devices), the state change of the IoT device is *reflected* on all user devices instantly (More details in section 5).

3.2 Technical Challenges

To enable FLUID-IoT workflow, we address two technical challenges: *C1) How to make UIs across different activities (i.e., app page) to be accessible and distributable for simultaneous multi-user interaction*; and *C2) how to distribute UIs selectively to multiple users while maintaining a responsive and consistent UI interaction (i.e., minimizing performance overhead).*

C1: In multi-user IoT environments, different users often want to control different devices. However, the control UIs for different devices are usually scattered across various activities within the IoT app, requiring constant switch between activities to, at best, sequentially handle each user interaction. To tackle this issue, we employ a multi-activity execution technique. This allows UIs across all activities to be rendered concurrently and handle user interactions simultaneously.

C2: Another important consideration is maintaining responsive and consistent UI interaction across all user devices. This is especially critical in the IoT landscape, where unsynchronized UIs may lead to unintended user actions due to misinterpretation of device states. For instance, a delay in updating the light bulb's status could lead a user to repeatedly pressing the power button, inadvertently

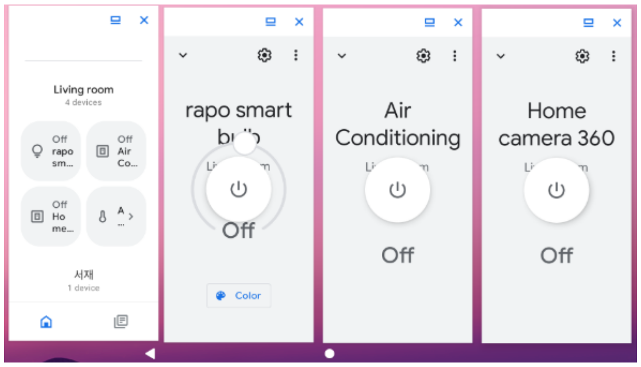


Figure 4: Google Home launched in multi-activity mode, allowing all its interfaces (i.e., activities) simultaneously available for concurrent user interactions.

turning off a bulb another user had already switched on. To address this challenge, we introduce *dynamic UI channel switching*, a method that uses multiple multicast channels to reduce synchronization lag and allows different sets of UIs to be distributed to different users based on their specific permissions.

3.3 Multi-Activity App Execution

To allow all control UIs to be simultaneously distributable and available for multi-user interaction, we enable the IoT app to run multiple activities on the foreground. In traditional mobile operating systems, apps are typically designed to run single activity in the foreground at a time. To extend such design, we leverage Android OS's `'Launch adjacent'` feature [19]. This feature enables apps to launch multiple activities side-by-side on the same screen, which is originally intended for developing a multi-window app targeted for large-screen or multi-screen devices. To integrate this feature into an existing IoT app, we inject a single line of code indicating that we want app's each activity to launch in a separate window. More specifically, we replace the following `startActivity` code:

```
Intent intent = new Intent(this, SecondActivity.class);
...
startActivity(intent);
```

with

```
Intent intent = new Intent(this, SecondActivity.class);
...
// Add flags to launch new activity in a new window.
intent.addFlags(Intent.FLAG_ACTIVITY_LAUNCH_ADJACENT |
     Intent.FLAG_ACTIVITY_NEW_TASK);
startActivity(intent);
```

This simple code injection or code modification allows app's all activities (i.e., all interfaces) to be simultaneously available for multi-user interactions.

3.4 UI Extraction and Individual Rendering

The next step in distributing UIs involves extracting the pixel data for each individual UI element. However, modern operating systems render collection of UIs (i.e., app screen) in a batch manner for the sake of resource efficiency, making it difficult to obtain individual UI elements' pixel data, especially when it is hidden behind other UIs (e.g., Figure 4).

For this reason, FLUID-IoT decouple the UI elements from the conventional batch-style rendering pipeline and renders each of them in an isolated frame buffer (Figure 5). Specifically, FLUID-IoT's *UI Extractor* first extracts the target UIs (i.e., the UI we want to distribute) from the app's UI Tree, a tree-shaped data structure that groups a collection of UIs to be rendered. UI elements in the same UI tree indicate that they should be drawn together on the same window (i.e., a frame buffer), and therefore subject to batch rendering. Conversely, this also implies that UIs in different UI trees can be rendered separately on different buffers.

FLUID-IoT's *UI Renderer* leverages this mechanism by creating multiple instances of an empty UI tree and assigning each extracted UI its own dedicated UI tree. Then, to provide UI trees with isolated frame buffers, *UI Renderer* creates an independent render space called Virtual Display [14] for each of the UI trees. The resolution of each Virtual Display is created equal to the size of the UI it is associated with. The operating systems' rendering pipeline will then automatically traverse through each UI tree independently and draw target UIs on their own dedicated frame buffers (i.e., Virtual Displays).

Our method of *extracting* the target UI from its original canvas (i.e., UI tree) and *redrawing* it in a separate window offers several advantages. Firstly, it ensures the complete visibility of the UI, even if it was originally hidden behind other elements. Secondly, FLUID-IoT is resistant to dark patterns, such as placing overlay ads over essential UIs. As long as the target UI's object exists within the operating system's UI tree, we can migrate it to a new window, thereby preventing it from being obscured or affected by these dark patterns.

3.5 Selective UI Streaming and Dynamic Channel Switching

Once we have access to the pixel data for each UI element, *UI Streamer* live-streams (i.e., distribute) the UIs to user devices. For each Virtual Display, we also generate an H.264 encoder that encodes the raw pixels drawn on the Virtual Display's frame buffer. Each encoded output stream is then be sent to the user devices through its dedicated network channel. The problem arises, however, because the UIs that each user device wants vary from user to user and from time to time. For instance, when the user wants to control the smart bulb, FLUID user app should display UIs associated with the smart bulb, and when the user wants to control the smart speaker, FLUID user app should switch to UIs associated with the smart speaker. In the interest of resource efficiency and network overhead, we cannot simply stream all UIs uniformly to every user device.

To solve this issue, *UI Streamer* dynamically switches the UI streaming channels based on which UIs should be displayed on which user devices (Figure 6). More specifically, in addition to the control permissions of each user, we also verify which IoT device the user is currently trying to control. Leveraging the fact that in most IoT apps, each IoT device has its own dedicated control activity, we reasonably assume that any user interaction that causes an activity transition is a switch in the IoT device that the user wishes

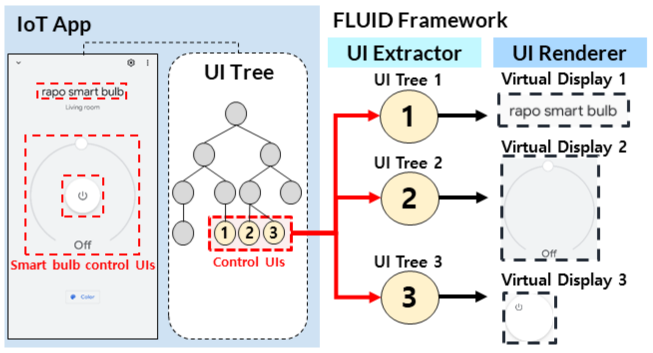


Figure 5: FLUID-IoT extracting and rendering each UI element in an isolated manner so that it can obtain pixel data of each individual UI element. The figure illustrates UI extraction and rendering of a single activity, but in the real system, the same process occurs simultaneously for every activity.

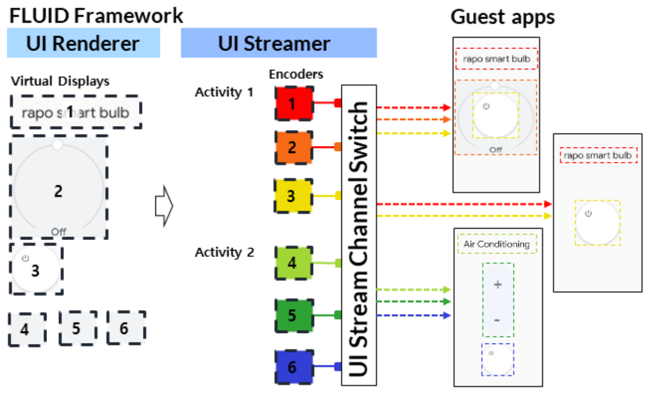


Figure 6: FLUID-IoT streaming UIs to FLUID user apps through channel switching. UI Streamer dynamically changes connected streaming channels to selectively distribute only the UIs that are permitted to the users.

to control. Therefore, each time the `startActivity()` function is called, FLUID-IoT first checks which user device triggered the function call, and identifies the UIs that belong to that activity. Among those UIs, *UI Streamer* streams only the UIs that are authorized to the user's device. This allows each user device to have a different set of UIs displayed on the FLUID user app, and to be able to swap between control activities, as though it is running a real IoT app.

This design choice makes implementing access control in FLUID-IoT remarkably straightforward. Essentially, it boils down the access control process to deciding which control UIs to distribute to which user. Furthermore, the ability to switch the UI stream channels on-the-fly allows FLUID-IoT to flexibly coordinate the control permissions of users. These characteristics empowers FLUID-IoT with the potential to implement a variety of access control mechanisms without interfering with the internals of the IoT app. (More details in section 4)

3.6 FLUID User App

The role of the FLUID user app is to render the streams of UI pixels and enable users to interact with these UIs. FLUID user app consists of three components: 1) *UI Renderer* that receives and renders the UI streams, 2) *UI Composer* that scales and layouts the UIs to fit the screen, and 3) *Input Handler* that hijacks and forwards the user interaction to the host device. As these components are not platform specific, FLUID user app can be developed for any mobile platforms including Android, iOS, and webApp.

The UI Renderer takes streams of UI pixels from the host device and displays them on the user's device. For each stream channel, it creates an output surface, and renders its corresponding UI element on the surface. These surfaces act as proxies for actual UI elements running on the host device, allowing users to remotely interact with the IoT app through their own devices.

The UI Composer adjusts the layout and scale of the displayed UI elements to fit the user's screen. It calculates the resolution difference between the host and user devices to scale each UI surface accordingly. Then, it arranges these elements based on their original hierarchical relationships, ensuring an intuitive and consistent UI layouts.

The Input Handler manages all touch events that occur on the UI surfaces displayed on the user's device. When a touch event happens, it captures the raw data, like x and y coordinates, and forwards this along with the UI element's ID to the host device. The host then injects the event directly into the targeted UI element based on the given UI ID. If multiple users attempt to interact with the same UI element simultaneously, the input injection is performed in a first-come, first-served manner.

3.7 Implementation

FLUID-IoT framework can be implemented in two forms: *1)* an application-level SDK that can be easily merged into IoT apps' source code with only a few lines of code insertion, and *2)* a system-level framework that can seamlessly extend IoT apps with FLUID-IoT features. Integrating these FLUID-IoT frameworks into an IoT app essentially requires two steps:*1)* launching the IoT app's activities in Multi-Activity mode and *2)* providing a list of UI elements related to IoT device control (e.g., device names, power buttons, volume controls). The process of extracting, rendering, and streaming the UI elements is done automatically by the FLUID-IoT framework.

3.7.1 Application-Level SDK Integration. For application-level FLUID-IoT SDK integration, developers are required to modify their apps to launch in Multi-Activity mode, which typically requires only a one or two-line code modification per activity. Additionally, developers need to manually specify which UI elements should be exposed to the FLUID-IoT SDK. This step, while manual, gives developers control over their apps by letting them decide which UI elements are sharable.

3.7.2 System-Level Framework Integration. The system-level framework (i.e., custom Android OS) offers a streamlined integration process, eliminating the need to modify the app's source code. The host user simply selects the IoT app they want to integrate with FLUID-IoT. The FLUID-IoT Framework then automatically launches all activities of the selected app in multi-activity mode, making its

UI elements available to the framework. This method is particularly beneficial because it seamlessly accommodates existing IoT apps without requiring any code changes.

3.7.3 Implementation Specifics for Google Home. In this study, given that prominent mobile IoT apps are not open source, we implemented the FLUID-IoT's system-level framework inside Android OS (Android Open Source Project v.11 [15]) and used off-the-shelf `Google Home` downloaded from the Google Play Store as a target IoT app. Although `Google Home` serves as our primary IoT app example, we found that other Android IoT apps like `Amazon Alexa` and `Samsung SmartThings` are similarity compatible (See subsection 5.1). In addition to the software integration, the FLUID-IoT system also requires a dedicated device to function as the host device. The host device can be any smart device capable of running IoT apps. In our implementation, we used a Google Pixel 4XL (released in 2019) as a host device and found that it can play the role of a host device flawlessly.

4 UTILIZATION OF FLUID-IOT

FLUID-IoT, combined with various access control mechanisms, opens up a broad set of applications, allowing users to configure their IoT environments in secure, finely-tuned, and adaptable ways. This section outlines the workflow of FLUID-IoT and introduces three different access control mechanisms and application areas to highlight the capability of FLUID-IoT.

4.1 Workflow

The FLUID-IoT system involves two primary stakeholders: the host user and the guest users. The host user, typically the owner of the IoT devices, is responsible for configuring and managing the access permissions within FLUID-IoT. Guest users, conversely, are those who interact with the IoT devices through the UIs shared by the host user.

Host user workflow. FLUID-IoT offers an intuitive interface for the host user to configure permission settings. As illustrated in Figure 7, FLUID-IoT displays all control UIs of the given IoT app alongside various access control mechanisms for configuration. Using these interfaces, the host user begins by selecting the specific UIs they wish to share. For each selected UI, they configure the permissions using different access control mechanisms. Once the permissions are set, the host user saves these configurations. The configurations are stored as an XML file, which includes a list of UI specifications (i.e., UI id) and its corresponding policy. However, this configuration file may become outdated if the IoT app undergoes a major update that changes the UI specifications. In such cases, the host user needs to re-configure the permission settings.

Guest user workflow. Guest users start by launching the FLUID-IoT user application on their devices, which serves as the gateway to accessing the shared UIs. To access these UIs, the FLUID-IoT user app needs to establish a connection with the host device. This can be achieved through various methods such as scanning a QR code provided by the host user or using a near-device search feature to automatically detect and connect to the host device. Upon successful connection, the guest user can see the UIs they have been granted access to. They can then interact with these UIs within the constraints of the permissions set by the host user.

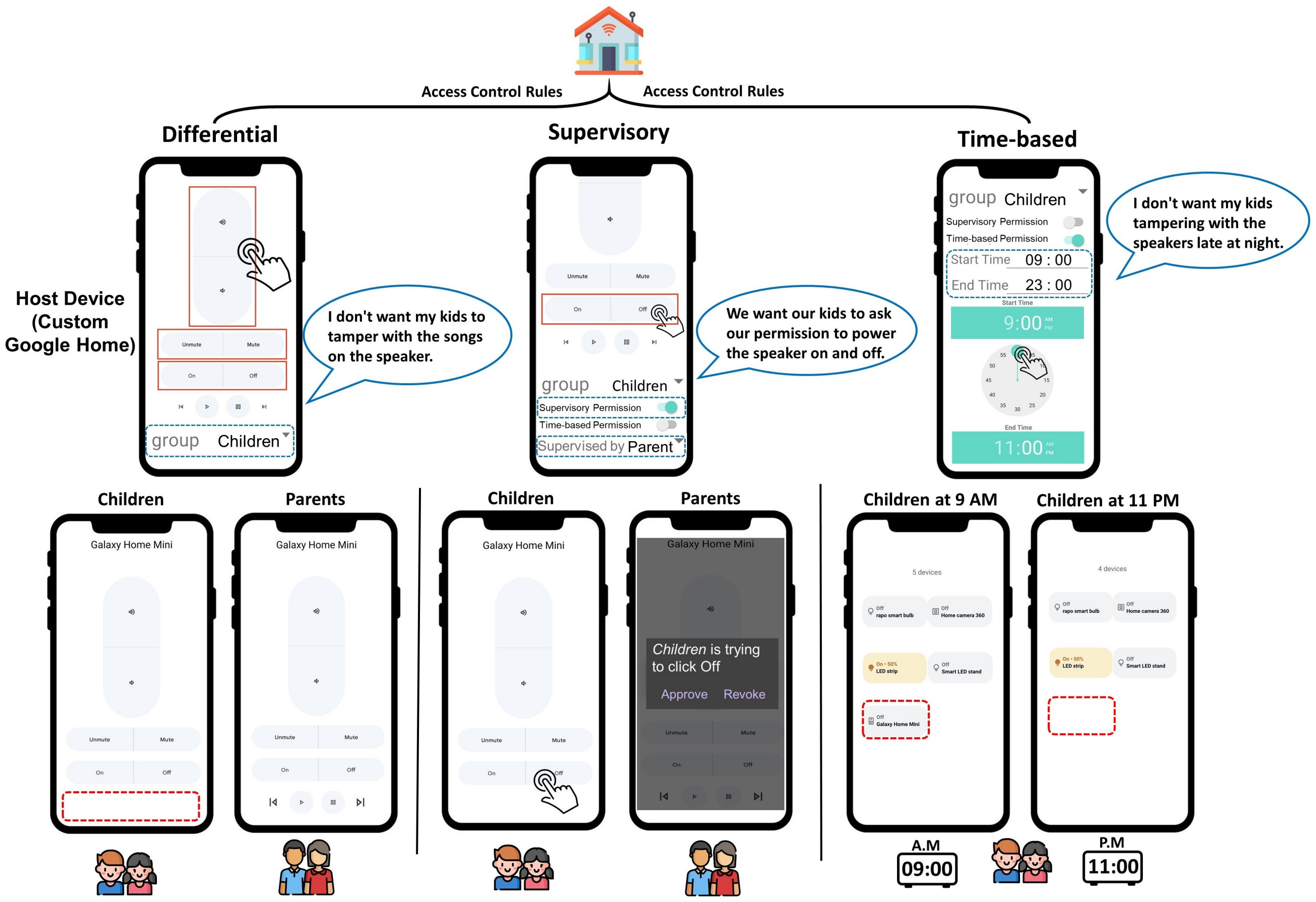
4.2 Access Control Mechanisms

In this paper, we have implemented three different access control mechanisms: *1) Differential access control*, *2) Time-based access control*, and *3) supervisory access control.* Note that these mechanisms are just a fraction of possible policies that FLUID-IoT can support. Any access control mechanisms imaginable can be implemented through FLUID-IoT framework with proper engineering. We demonstrate each mechanisms using the off-the-shelf `Google Home` application (Figure 7). Throughout the rest of the paper, we refer to Google Home retrofitted with three access control mechanisms, as *"custom Google Home".*

4.2.1 Differential access control.
Definition. Differential access control is a type of access control mechanism that grants different levels of access to different users or groups based on their roles, responsibilities, and privileges. In FLUID-IoT system, user permission can be differentiated not only by the types of devices, but also by the range of actions that users can perform on each device. For example, while you can grant both User A and User B access to a smart light bulb, you can also configure the permissions so that User A can control both brightness and the power of the light bulb, while User B can only access the power.
Implementation. To implement differential access control with granularity down to the UI level, we developed an interface that allows a master user (i.e., a user who owns the host device) to configure which users or user groups are permitted with which UI elements (Figure 7 (a)). The configuration is saved as a JSON file, and the FLUID-IoT framework reads it to set up the UI streaming channels between each user device and the host device.

4.2.2 Time-based access control.
Definition. Time-based access control is a type of access control mechanism that grants access only for a limited period of time. Time-based access control can be useful in situations where organizations need to ensure that access is automatically revoked once the authorized time period has ended. This reduces the need for manual intervention and potential errors or oversights in revoking access.
Implementation. To implement time-based access control, we added an interface that allows the master user to specify the time period during which a user can control the IoT device (Figure 7 (c)). The time period is saved along with the differential access control configuration, and FLUID-IoT periodically monitors the time and immediately disconnects the UI streaming channel when the time period ends.

4.2.3 Supervisory access control.
Definition. Supervisory access control is a mechanism that involves the active monitoring and supervision of users who are granted access to IoT devices. In supervisory access control, authorized individuals monitor the activities of other users to ensure that they are using the IoT devices in accordance with the policies and rules. In FLUID-IoT system, all control actions performed by the supervised user are reported to the authorized user. The supervised user's actions are executed only upon receiving consent from the authorized user. For instance, if a child tries to switch the TV channel to a late-night movie, a notification is sent to the parent's



(a) Diffential Permission (b) Supervisory Permission (c) Time-based Permission

Figure 7: FLUID-IoT access control interfaces built on top of Google Home. (a) Differential access control allows the host to grant different levels of access to users with granularity down to the UI level. (b) Supervisory permission allows the host to configure supervisory permission to monitor other users' actions over the IoT device. (c) Time-based access control allows users to give permission only for a limited period of time.

FLUID user app. Only after the parent grants consent through the app will the channel actually change.

Implementation. To implement supervisory access control, we designed an interface where a master-user can designate specific user supervision relationships (Figure 7 (b)). The relationships are saved along with other access control configurations. Based on the established relationship, the FLUID-IoT framework directs the supervised users' FLUID user app to forward their input events to the authorized users' device rather than to the host device. When the authorized user application receives an input event, it prompts a consent message, informing what control action the supervised user is trying to do. The authorized user can then approve or revoke the action. Upon approval, the input event gets forwarded to the host device for execution.

4.2.4 Mixing multiple mechanisms. One distinct feature of FLUID-IoT is that it allows access control mechanisms to operate at the granularity of the UI level, meaning that control permissions can be applied differently not only for each IoT device but also for each feature within the device. For instance, the system can be configured to allow a user full-time access to the light bulb's power control, but temporary access to the brightness control.

In addition, control mechanisms can work in parallel to combine multiple access control policies. For example, Time-based access control and supervisory access control can be used in conjunction to require the administrator's consent when controlling an IoT device during specified hours.

5 PERFORMANCE EVALUATION

We evaluated FLUID-IoT in terms of coverage, latency, and memory consumption to verify whether FLUID-IoT has sufficient performance to be used in our everyday IoT settings. Across the evaluation, a Pixel 4 XL (AOSP v.11) was used as the FLUID-IoT host device, and one Pixel 4 XL (Android v.11) and one Pixel 6(Android v.13) were used as guest devices.

5.1 Coverage

To assess the compatibility of the FLUID-IoT framework with existing IoT platforms, we conducted a coverage test on the eight most downloaded IoT apps available on Google Play. Utilizing the modified Android OS outlined in subsection 3.7, we examined whether the FLUID-IoT framework could successfully distribute the UI elements for each of these off-the-shelf IoT apps. The types of UIs utilized in these apps include device name fields, device state fields, a range of buttons (such as power, up/down, and mute), and sliders. Note that the type of UI does not influence the FLUID-IoT's coverage, because FLUID-IoT manipulates UIs at their object level (i.e., a code-level entity). As long as FLUID-IoT can access the target UI's object, it can distribute UIs to other devices.

As demonstrated in Table 1, the current implementation of FLUID-IoT successfully supports six out of the eight IoT apps, which accounts for nearly 90% of the total IoT ecosystem. In the case of `LG ThinQ`, testing was not feasable due to a non-systematic issue that prevents it from running on rooted devices (e.g., AOSP). Nevertheless, our static analysis of the app found no technical barriers to supporting `LG ThinQ`, as it meets the requirements outlined in subsection 3.7. Conversely, FLUID-IoT is incompatible with `Philips Hue` because it employs WebView, a third-party UI library that prevents FLUID-IoT from accessing its UIs. Overall, FLUID-IoT is highly compatible with existing IoT platforms and it can cover the majority of the IoT ecosystem. The minute coverage hole can be addressed through proper implementation (see subsubsection 7.1.1).

5.2 Latency

To evaluate FLUID-IoT's performance from the users' perspective, we measured two representative latencies (Response time and synchronization latency) that users would experience while using the FLUID user app, and compared them to those of the original `Google Home` app. The experimental setup included one smart bulb, one FLUID-IoT host device running a custom `Google Home`, and two user devices running either the FLUID-IoT user app or the original `Google Home` app. We recorded the screens of the two user devices at a 120Hz refresh rate to measure the latency. The latency measurements have an error bound of ±8.3ms since each video frame represents an 8.3ms time slot. Both the custom `Google Home` and the original `Google Home` use Wifi protocol to communicate with the IoT devices (e.g., smart bulb). All devices were connected to the same Wi-Fi access point with a 140 Mbps connection and an average round-trip time (RTT) of 35.92ms with a standard deviation of 69.60ms. We repeated each experiment 20 times.

Response Time. We first measured the response time. Response time is defined as the time it takes for the result of a user action (e.g., tapping the power button) to be visually reflected back in the app. Figure 8(a) compares the response time of FLUID-IoT and the original Google Home.

The average response time of the original `Google Home` was 16.05 ms (median=17; stdev=3.818), which is one Vsync time of smartphones (e.g., Pixel 4XL, Pixel 6) with a screen refresh rate of 60-90Hz. This low latency is achievable because most IoT apps use client-side prediction to provide a seemingly instant reaction. In other words, instead of waiting for the IoT device to broadcast its updated state, IoT apps compute and predict the outcome of the user's action locally and display the *predicted* result.

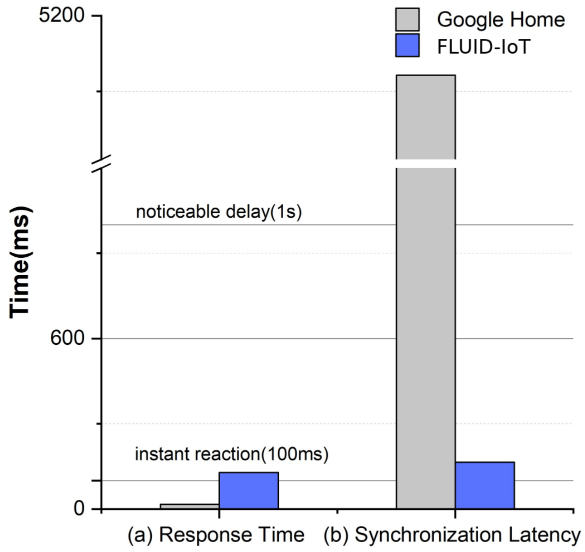


Figure 8: Response Time and Synchronization Latency of the Google Home and FLUID-IoT in milliseconds. While FLUID-IoT exhibits a somewhat higher response time, it remains close to the generally accepted threshold for users to perceive an interface as "instant". More significantly, FLUID-IoT's synchronization latency is markedly lower, offering a dramatically improved user experience in multi-user settings.

However, in multi-user settings, where predictions are made concurrently and scattered across multiple devices, it can result in inconsistent app behavior. For instance, we observed that when we press the power button of the smart bulb simultaneously on two different `Google Home` apps, both apps displayed that the bulb is on, even though it is actually off. In some cases, one of the two actions was completely ignored, and the bulb did not turn off as we intended.

On the other hand, FLUID-IoT exhibits a higher average latency of 128.619 ms, with a median of 101 ms and a standard deviation of 66.025 ms. This increase in response time is due to the network overhead of forwarding touch events to the host device and retrieving the outcome in the form of pixels. Yet, considering that response times of 100 ms are generally regarded as the threshold for users perceiving an interface as "instant" [33], the response time of FLUID-IoT can still be considered acceptable for most use cases. Additionally, since the state prediction only occurs within the host device in a sequential manner, FLUID-IoT ensures consistent behavior even in multi-user environment.

Synchronization Latency. We then evaluated the synchronization latency under multi-user settings. We define synchronization latency as the time duration required for *all* IoT apps running on each user device to synchronize with the updated state of the IoT device. Figure 8(b) compares the synchronization latency of FLUID-IoT and that of the original Google Home. The figure shows that the original `Google Home` took an average of 5121.45 ms (median=5070.5; stdev=559.55). This implies that when user A turns on the smart bulb through her `Google Home` app, it takes about

App Name	# of Downloads	FLUID-IoT	Failure Causes
Smart Things	500M+	O	-
Google Home	100M+	O	-
Alexa	100M+	O	-
LG ThinQ	50M+	△	Not testable on rooted device, but theoretically applicable
Mi Home	10M+	O	-
Smart Life	10M+	O	-
TP-Link Tapo	5M+	O	-
Philips Hue	5M+	X	Using WebView to draw UIs

Table 1: Coverage Test for eight IoT applications. 'O' and '△' and 'X' indicate that FLUID-IoT is applicable, FLUID-IoT is possibly applicable, and FLUID-IoT is non-applicable, respectively. FLUID-IoT successfully supports six out of the eight prominent IoT apps, demonstrating that it is highly compatible with existing IoT platforms and it can cover the majority of the IoT ecosystem.

Device Types	Air Conditioner	Bulb	Speaker	Total
# of UIs	3	4	5	12
# of pixel	1,188,786	1,488,240	1,780,884	4,457,910
Memory(MB)	4.76	5.95	7.12	17.83

Table 2: FLUID-IoT's additional memory consumption for distributing UIs of three different IoT devices. The total extra memory consumption (17.83 MB) accounts for just 0.297% of the average RAM in modern smartphones. This indicates a minimal impact on device performance.

5 seconds for user B's Google Home app to update and show that the bulb has been turned on. This can significantly degrade the user experience, as it may mislead users to perform unintentional actions based on the outdated device state. This problem arises because an an app cannot "predict" state changes triggered by other devices. Instead, it must wait for the IoT device to broadcast its updated state—an event that occurs approximately every 5 seconds in the case of Google Home. It is important to note that broadcast intervals vary between different IoT platforms. Hence, different IoT platforms may exhibit different synchronization delay.

On the other hand, FLUID-IoT demonstrates an average synchronization latency of 165 ms (median=134; stdev=85.269), which is substantially lower than that of the original Google Home and slightly higher than the response time. We attribute this to our centralized architecture that shares one IoT app across multiple user devices. Any changes to the IoT app's UI are propagated simultaneously to all user devices. As a result, when user A turns on the smart bulb, all users are instantly notified of the change.

In summary, when compared to the original behavior of the Google Home, FLUID-IoT and its custom Google Home increases the response time from 16.05 ms to 128.619 ms, while reducing the synchronization latency from 5121.45 ms to 165 ms. Although there is an increase in response time that might or might not be noticeable to the users, the substantial reduction in synchronization latency can bring a significant improvement to the user experience under multi-user settings. Moreover, considering that FLUID-IoT ensures consistent control behavior throughout the single- and multi-user environments, we can conclude that the benefits of FLUID-IoT in multi-user IoT environment outweigh the loss.

5.3 Memory Consumption

FLUID-IoT consumes extra memory space when creating virtual displays (i.e., frame buffers) for the UI distribution. And FLUID-IoT creates a new virtual display for each new UI to distribute. We measured how much memory FLUID-IoT consumes for each additional virtual display. We used the Android Debug Bridge command `dumpsys meminfo` to measure the memory usage of FLUID-IoT while distributing 12 different Google Home UIs from three IoT devices—Air conditioner, smart bulb, and smart speaker.

Table 3 outlines the average memory consumption for each UI type. As a result, we found that the extra memory consumption for each UI exactly matches the amount of memory required to render the UI in the ARGB8888 format (4 bytes per pixel). This indicates that the memory consumption of FLUID-IoT scales linearly with the total pixel count of the UIs distributed. Furthermore, Table 2 details the memory usage of each individual IoT device. When distributing UIs of all three IoT devices, FLUID-IoT consumed 17.83 MB of extra memory. This accounts for only 0.297% of the average RAM size of modern-day smartphones, which is approximately 6 GB [42]. These results suggest that FLUID-IoT's memory consumption is modest and unlikely to significantly impact the performance of most Android smartphones, even when working with multiple UIs and multiple IoT devices.

6 DEMONSTRATION OF USABILITY

To better understand the effectiveness and usefulness of FLUID-IoT in terms of user experience, we conducted a lab study with our custom Google Home. In each session, each participant acted as a host of a multi-user IoT environment and completed permission-setting tasks for six distinct scenarios. More specifically, the study aimed to answer the following research questions: (1) *Is FLUID-IoT's novel approach to implement access controls on top of existing IoT platforms effective?* and (2) *Is FLUID-IoT's unique ability to offer granular access control over individual device functionalities useful?*

6.1 Participants

We recruited 17 participants (8 females, 9 males, ages 19 to 29 with avg=24.41 and std=3.18) who were recruited through online school communities and local community groups. Each study session lasted 40 to 60 minutes, and we compensated each participant $10 for their time. The recruitment and the experiments were in accordance with our institution's IRB policies and the consent forms.

6.2 Study Procedure

At the beginning of each session, participants were given a brief tutorial on how to use the custom Google Home. In the tutorial, they

UI Types	Power Button	Up/down Button	Dual Button	Device Name	Device State	Slider
# of pixels (Avg)	187,152	926,100	173,040	113,787	29,580	1,238,400
Memory (Avg)	0.75 MB	3.70 MB	0.69 MB	0.46 MB	0.12 MB	4.95 MB

Table 3: Average memory consumption depending on UI type.

were demonstrated how to use the three different access control mechanisms: *i)* differential access control, *ii)* time-based access control, and *iii)* supervisory access control. Following the tutorial, participants were given three Google Pixel 4XL phones: One host device with custom `Google Home` installed, and two user devices with FLUID user app installed.

Then, participants received six multi-user IoT scenarios, in which they were asked to perform given permission-setting tasks. After completing all tasks, participants were asked to evaluate the experience using the System Usability Scale (SUS) [8] and report the perceived usefulness of each access control mechanism in a post-survey.

6.3 Multi-User IoT Tasks

Our study is divided into two phases: warm-up tasks (T1~T4) and complex tasks (C1 and C2). The warm-up tasks were designed to familiarize participants with the multi-user IoT environment and the FLUID-IoT. During this phase, participants were encouraged to ask questions. On the other hand, the complex tasks aimed to challenge participants and assess the usability of the FLUID-IoT in more demanding situations. During the complex tasks, participants were not allowed to ask any questions and were encouraged to keep trying until they successfully completed the tasks by themselves.

Throughout the tasks, participants role-played as Airbnb hosts and managed access permissions for guests within a multi-user IoT environment. In the warm-up stage, participants explored distinct access control mechanisms—differential access control, (T1 and T2), time-based access control (T3), and supervisory access control (T4)—within unique scenarios. For the complex tasks, participants were challenged to blend these mechanisms to create a balanced and effective access control strategy for more demanding scenarios. The detailed scenarios and task descriptions are included in our Appendix.

6.4 Results and Findings

6.4.1 Pre-survey. In the pre-survey, 15 out of 17 participants had prior experience using IoT technology, and 10 of those 15 also had experience sharing IoT devices. They had experience sharing IoT devices mostly through a voice-assisted smart home hub or using other user's smartphone who has permission to access the IoT devices. However, these IoT sharing methods poses security risks, as they cannot adequately protect device ownership and user privacy.

6.4.2 Evaluation on tasks performance. All participants completed the warm-up tasks successfully. They went through each task step by step, understanding the system by seeing how each setting was reflected on user devices in real-time. For the subsequent complex task, all but two participants were able to complete it without any

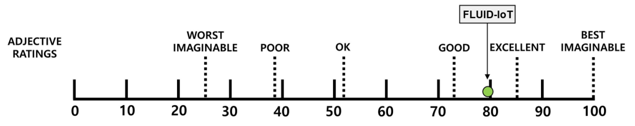


Figure 9: In the Adjective Rating Scale introduced by A.Bangor et al. [6], FLUID-IoT is ranked *good* usability.

assistance. One participant made a minor mistake due to misreading the instructions, while another participant made a trivial error but promptly corrected it with slight intervention. Consistent with the high task completion rate, the average score on the SUS question *"I would imagine that most people would learn to use this system very quickly"* was 5.94 out of 7, indicating that the access control interface of FLUID-IoT is easy to learn.

6.4.3 System Usability Scale. Overall, participants felt FLUID-IoT has high usability. On the 7-point System Usability Scale (SUS), FLUID-IoT scored on average 79.08. Figure 9 shows where our average score of SUS (79.08) is ranked in user-perceived usability ratings. It suggests that our system has a relatively high level of usability.

We attribute this high rating to FLUID-IoT's innovative design, which allows for the flexible implementation of access controls over existing IoT applications. Specifically, the system scored a highest 6.41 on the question, *"I found the various functions in this system were well integrated,"* and a lowest 1.64 on the question, *"I thought there was too much inconsistency in this system."* These results indicate that FLUID-IoT effectively and seamlessly integrates its access controls mechanisms as though they are built-in features of existing IoT platforms

6.4.4 Feedback on the Access Control Mechanisms. Following the overall system usability scoring, we asked participants to rate the usefulness of three access control mechanisms on a 7-point Likert scale. The results are as follows: Differential permission scored 6.17 on average, Time-based permission scored 6.29 on average, and Supervisory permission scored 5.41 on average. Below, we share some interesting comments and observations from the participants.
Differential Permission: Participants expressed great satisfaction with the ability to control permissions for each fine-grained functionality of a device. A significant number of participants (9 out of 17) gave it the top score (7) and advocated that IoT devices should have a function-level permission settings—"at first glance, it looks complicated, but there certainly are circumstances that need such fine-grained level of IoT sharing (P4)."

Some notable feedback are: *"It was convenient to see clearly what is being shared and what is not (P14)"*, and *"It is similar to the real remote controller so it will be familiar even to the first-time users (P9)."* These comments show that our implementation of access control interfaces built inside the Google Home app (Figure 7) makes access control easy and intuitive.

Time-based Permission: Among three representative access control mechanisms, time-based access control received the highest score. Many participants agreed that granting short-term authorization would be useful in various domains, such as accommodations, smart homes, and many other public places. Some comments are as follows: *"It will be very useful for shared living spaces or households with children when they want to enforce mandatory lights-out time (P2)"*, *"It will be useful for cases like AirBnB where guests change frequently (P5)"*, and *"It will be useful for any public places (P12)."* Some participants also suggested that it will be even more useful if guests are notified of their authorized time period. P6: *"I think it is similar to making a reservation, but it would be better if guests can know at which time period, they can access the device."*

Supervisory Permission: Opinions on Supervisory Permission were somewhat divided. Some participants thought having to request or accept authorization every time there is an action would be cumbersome, while others felt it is a necessary feature despite its inconvenience. P2 commented that *"Seems like it would be cumbersome for both host and guests to have to request/accept each time."*, while P1 commented *"Having to ask for permission can be a hassle, but I think it's useful in preventing abusive use."* To address the inconvenience, we could consider authorizing the user for a certain period of time after obtaining permission once or using password protection, as P16 commented *"...or you can use password to lock the permission..., so that hosts can adjust the level of security."*

User preferences: Participants were also asked to choose the most useful access control mechanism from three. Notably, while Time-based access control had the highest average usability score, edging out Differential access control by 0.12 points, when asked to directly compare the three, 47.1% of participants favored Differential access control, followed by Time-based at 29.4% and Supervisory at 23.5%. This indicates that FLUID-IoT's unique ability to control access to specific functionality of a device is the most useful and desired approach in a general day-to-day environment.

6.4.5 *Implications from the User Study.*

Usefulness as a guest: While we designed FLUID-IoT with a focus on how to make access control easier and intuitive for the hosts (i.e., owner of the devices or a shared place), many participants commented on its usefulness for the guests as well. On the survey question, *"As a guest, is displaying only the buttons that you can control useful?"*, participants scored an average score of 6 out of 7—*"I do not need to think about much. If I can see it, I can control it (P2)"*.

In addition, there were some interesting comments on psychological backlash when displayed with UIs that they cannot control. P3 commented that *"if there is a button that I can't control, it is just a waste of information, and it can incur inappropriate desire to violate privileges."* P9 and P11 also commented *"People tend to desire things that they are not allowed"* and *"I get annoyed when there are buttons that I can't use,"* respectively. This not only highlights the benefits of FLUID-IoT utilizing UI distribution technique, but also exposes the shortcomings of prior works [40, 41], which uses input blocking to implement access control.

IoT in `Airbnb`: Participants found FLUID-IoT particularly useful for short-term rentals, with 14 out of 17 participants rating FLUID-IoT as *"very useful"* in the AirBnB scenarios (T1∼T4). This aligns with the growing interest in smart devices and related policies within the `Airbnb` community [2, 22]. According to a recent study on smart home device use by Airbnb hosts [12], Airbnb hosts want special access control settings that would allow them to share the capability of smart home devices with their guests without compromising their own privacy and/or security.

Furthermore, some hosts reportedly wished for IoT devices to be able to synchronize with the Airbnb account so that guests do not have to manually install IoT apps. FLUID-IoT can be a very useful solution to this demand, as FLUID user app can interact with any IoT platform, and guests do not need to install IoT apps on their local devices. This demonstrates that our research is a relevant and timely study that can address current challenges related to smart device usage in short-term rentals.

Insights for future work: Many participants shared valuable feedback on how FLUID-IoT can be improved. P4 commented, *"It's unfortunate that the system works only within the same Wi-Fi AP. It would be much better if I could control permissions outside of the home."* The current implementation of FLUID-IoT has not considered the need for remote access control. To address this issue, we can host a web server inside the host device so that users can remotely interact with FLUID-IoT's access control interface.

Another insightful comment by P11 was, *"In some cases, people will use physical buttons on the device. If we can't prevent them from physically controlling the device, we can make the system automatically revert their action."* When access control within a digital space can be ineffective in some physical environments, the suggestion by P11 could be a solution to control access permissions without modifying the physical buttons.

7 DISCUSSION

In this section, we discuss the limitations of our current implementation and how FLUID-IoT can be improved through future research.

7.1 Limitations

7.1.1 *Unsupported IoT apps.*

The current implementation of FLUID-IoT operates exclusively with Android apps that use the native Android UI library. It does not support IoT apps employing third-party UI engines (e.g., Flutter [17], React Native [32], WebView) or iOS applications (e.g., HomeKit) because FLUID-IoT's UI distribution mechanism requires access to an app's UI elements, but the UIs in these apps are encapsulated within closed-source UI engines. Nonetheless, the overarching design of FLUID-IoT is compatible with other mobile UI engines. FLUID-IoT is built on the principle that UIs are organized in a hierarchical structure (i.e., UI tree) and can be displayed on a separate frame buffer (e.g., Android's Virtual Display or iOS's UIScreen). This approach aligns with the common design paradigm of GUI-based systems. Therefore, despite its current limitations, FLUID-IoT holds potential for broader application in various mobile environments.

7.1.2 *Potential threat models.*

While FLUID-IoT is effective in integrating access control into GUI-based device control, potential threat models exist that can bypass or neutralize FLUID-IoT.

Device control through other modalities. Users can circumvent FLUID-IoT by using alternative device control modalities such as voice-activated smart assistants (e.g., Google Assistant and Alexa) or physical buttons. Although addressing all possible modalities falls outside this study's scope, potential workarounds exist, such as employing Google's voice and face match features for voice-operated devices or using protective covers on physical buttons. As the landscape of IoT controls continues to diversify, embracing diverse modalities like voice, Near-Field Communication (NFC), and vision, future research is needed to enhance the FLUID-IoT framework to support a more inclusive and multimodal access control system.

Malicious apps. FLUID-IoT's design of access control is based on the principle that users can only control the device through the UIs that are explicitly shared with them. However, this design can be compromised if a user employs a malicious app that simulates "interactions" with UI elements that were not actually shared with that user. To counteract such malicious interactions, FLUID-IoT's host device verifies the validity of incoming user inputs by ensuring that the UI element targeted by the input is indeed permitted on the guest device from which the input originated.

7.1.3 *Single point of failure.*

FLUID-IoT system relies on a host device to act as an intermediary between IoT devices and users, responsible for all computation and communication within the shared IoT environment. As a result, FLUID-IoT has a risk of a single point of failure; if the host device experiences downtime or failure, all FLUID user apps would cease to function as well. To mitigate this, FLUID user app can be engineered to automatically launch the IoT app and run it locally when the connection to the host device is lost. While access control may not function correctly until the host device is restored, users would still retain control over IoT devices. Other fault handling techniques, such as implementing a backup host device, and improving system robustness and fault tolerance can also be explored to further mitigate the risk.

7.2 Future Works

7.2.1 *More access control mechanisms.*

As previously highlighted, FLUID-IoT can offer a range of access control mechanisms beyond those implemented in this paper. Possible examples include a *consensus-based control mechanism* that utilizes ballots to manage public IoT devices, a *location-based access control* that adjusts control permissions or priority based on the user's proximity or position relative to the device, and a *rate-limiting access control* that limits the number of actions that each user can perform within a specified time period.

These mechanisms can be freely implemented inside the FLUID-IoT framework by designing algorithms that determine where each UI should be distributed and how user inputs should be handled. In future work, to further enhance user experience, we could provide a policy generation language or interface that allows end-users to create their own access control mechanisms.

7.2.2 *Massive user/IoT environment.*

Despite its low level of performance overhead, FLUID-IoT may face scalability challenges as the number of users and IoT devices grows at a massive scale. The task of encoding and distributing a large number of UI elements could impose significant overhead on the host device and affect its performance. However, considering that the number of users and IoT devices in our everyday environment has yet to reach a massive scale [43], FLUID-IoT can sufficiently handle the majority of shared IoT scenarios, including those found in smart homes and smart offices. To address future scalability issues, optimization techniques such as streaming UIs in lower resolution, destroying the frame buffer and encoder of inactive IoT devices, or designing a dedicated server device, are viable avenues for future research.

7.2.3 *Combining UIs Across Multiple IoT apps.*

The current implementation of FLUID-IoT focuses on providing access control for a single IoT app at a time, ideally suited for users who operate within a singular IoT ecosystem. However, the fragmentation of IoT ecosystems typically demands the use of multiple, vendor-specific apps to achieve full control over all devices. While unifying protocols like Matter [36] exist, they do not provide complete support for all devices and functionalities, causing users to depend on multiple IoT apps.

To mitigate this issue, FLUID-IoT could be extended to accommodate UIs from multiple IoT apps, providing a more a more unified and streamlined user experience. One feasible approach is the integration of existing multi-app execution frameworks like *A-Mash* [27], which specializes in UI mashups across different applications. By incorporating such functionalities, FLUID-IoT could serve as a central hub for multi-app IoT control. This addition would allow FLUID-IoT to bridge the gap between disparate IoT ecosystems, streamlining user experience and setting the stage for a more unified and secure multi-app IoT environment.

8 CONCLUSION

The current landscape of shared IoT environments offers naive and simplistic access control measures. In this paper, we present FLUID-IoT, a framework that retrofits existing IoT platforms with *fine-grained* and *flexible* multi-user access control. FLUID-IoT uses a novel multi-user UI distribution technique to effectively control access to IoT devices by selectively distributing control UIs to users based on access control policies. FLUID-IoT not only allows users to flexibly implement access controls on top of existing IoT platforms without modifying their underlying IoT stacks but also enables fine-grained access control over individual device functionalities. Our evaluation of FLUID-IoT showcases its effectiveness in terms of performance, coverage, and usability. We believe that FLUID-IoT is a concrete step towards a more capable and safer IoT environment.

ACKNOWLEDGMENTS

This work was supported in part by the National Research Foundation of Korea (NRF) grant funded by the Ministry of Science and ICT (NRF-2022R1C1C1003123).

REFERENCES

[1] Habtamu Abie and Ilangko Balasingham. 2012. Risk-Based Adaptive Security for Smart IoT in EHealth. In *Proceedings of the 7th International Conference on Body Area Networks* (Oslo, Norway) *(BodyNets '12)*. ICST (Institute for Computer Sciences, Social-Informatics and Telecommunications Engineering), Brussels, BEL, 269–275.

[2] Airbnb. 2023. *Use of cameras and recording devices*. Airbnb, Inc. https://www.airbnb.co.kr/help/article/3061

[3] Apple. 2023. *Car keys and CarPlay. A smarter ride from start to finish.* Retrieved Mar 3, 2023 from https://www.apple.com/ios/carplay/

[4] Apple. 2023. *Use AirPlay to stream video or mirror the screen of your iPhone or iPad.* Retrieved Mar 3, 2023 from https://support.apple.com/en-us/HT204289

[5] Martin Armstrong. 2022. *Smart home devices to boom over the next 5 years.* WORLD ECONOMIC FORUM. Retrieved Apr 29, 2022 from https://www.weforum.org/agenda/2022/04/homes-smart-tech-market

[6] Aaron Bangor, Philip Kortum, and James Miller. 2009. Determining what individual SUS scores mean: Adding an adjective rating scale. *Journal of usability studies* 4, 3 (2009), 114–123.

[7] Mathieu Boussard, Dinh Thai Bui, Richard Douville, Pascal Justen, Nicolas Le Sauze, Pierre Peloso, Frederik Vandeputte, and Vincent Verdot. 2018. Future Spaces: Reinventing the Home Network for Better Security and Automation in the IoT Era. *Sensors* 18, 9 (Sep 2018), 2986. https://doi.org/10.3390/s18092986

[8] John Brooke. 1995. SUS: A quick and dirty usability scale. *Usability Eval. Ind.* 189 (11 1995).

[9] A.J. Brush, Jaeyeon Jung, Ratul Mahajan, and Frank Martinez. 2013. Digital neighborhood watch: investigating the sharing of camera data amongst neighbors. 693–700. https://doi.org/10.1145/2441776.2441853

[10] Camille Cobb, Sruti Bhagavatula, Kalil Anderson Garrett, Alison Hoffman, Varun Rao, and Lujo Bauer. 2021. "I would have to evaluate their objections": Privacy tensions between smart home device owners and incidental users. *Proceedings on Privacy Enhancing Technologies* 2021, 4 (2021), 54–75.

[11] Joanna F. DeFranco and Mohamad Kassab. 2021. Smart Home Research Themes: An Analysis and Taxonomy. *Procedia Computer Science* 185 (2021), 91–100. https://doi.org/10.1016/j.procs.2021.05.010 Big Data, IoT, and AI for a Smarter Future.

[12] Rajib Dey, Sayma Sultana, Afsaneh Razi, and Pamela J. Wisniewski. 2020. Exploring Smart Home Device Use by Airbnb Hosts. In *Extended Abstracts of the 2020 CHI Conference on Human Factors in Computing Systems* (Honolulu, HI, USA) *(CHI EA '20)*. Association for Computing Machinery, New York, NY, USA, 1–8. https://doi.org/10.1145/3334480.3382900

[13] Christine Geeng and Franziska Roesner. 2019. Who's In Control? Interactions In Multi-User Smart Homes. In *Proceedings of the 2019 CHI Conference on Human Factors in Computing Systems* (Glasgow, Scotland Uk) *(CHI '19)*. Association for Computing Machinery, New York, NY, USA, 1–13. https://doi.org/10.1145/3290605.3300498

[14] Google. [n.d.]. *VirtualDisplay.* Retrieved April 4, 2023 from https://developer.android.com/reference/android/hardware/display/VirtualDisplay

[15] Google. 2023. *Android Open Source Project.* Retrieved Mar 3, 2023 from https://source.android.com//

[16] Google. 2023. *androidauto.* Retrieved Mar 3, 2023 from https://www.android.com/auto/

[17] Google. 2023. *Flutter.* Google. Retrieved March 06, 2023 from https://flutter.dev/

[18] Google. 2023. *Google Cast SDK.* Retrieved Mar 3, 2023 from https://developers.google.com/cast

[19] Google. 2023. *Multi-window support.* Google. Retrieved March 06, 2023 from https://developer.android.com/guide/topics/large-screens/multi-window-support#launch

[20] Weijia He, Maximilian Golla, Roshni Padhi, Jordan Ofek, Markus Dürmuth, Earlence Fernandes, and Blase Ur. 2018. Rethinking Access Control and Authentication for the Home Internet of Things (IoT).. In *USENIX Security Symposium.* 255–272.

[21] Google Home. 2023. *Share a home and devices in the Google Home app.* Google. Retrieved Jan 17, 2023 from https://support.google.com/assistant/answer/9155535?hl=en-GB

[22] Lifty Life Hospitality. 2021. *Airbnb Smart Home | The Best Airbnb Smart Devices | 2021.* LIFTYLIFE. https://www.liftylife.ca/airbnb-smart-home-and-devices/

[23] Vincent Hu, David Ferraiolo, D. Kuhn, A. Schnitzer, Knox Sandlin, R. Miller, and Karen Scarfone. 2014. Guide to attribute based access control (ABAC) definition and considerations. *National Institute of Standards and Technology Special Publication* (01 2014), 162–800.

[24] Yue Huang, Borke Obada-Obieh, and Konstantin (Kosta) Beznosov. 2020. Amazon vs. My Brother: How Users of Shared Smart Speakers Perceive and Cope with Privacy Risks. In *Proceedings of the 2020 CHI Conference on Human Factors in Computing Systems* (Honolulu, HI, USA) *(CHI '20)*. Association for Computing Machinery, New York, NY, USA, 1–13. https://doi.org/10.1145/3313831.3376529

[25] William Jang, Adil Chhabra, and Aarathi Prasad. 2017. Enabling Multi-User Controls in Smart Home Devices. In *Proceedings of the 2017 Workshop on Internet of Things Security and Privacy* (Dallas, Texas, USA) *(IoTS&P '17)*. Association for Computing Machinery, New York, NY, USA, 49–54. https://doi.org/10.1145/3139937.3139941

[26] Vineet K. 2021. *Smart Office Market Size, Share&Analysis.* Allied Market Research. Retrieved Sep 06, 2021 from https://www.alliedmarketresearch.com/smart-office-market-A13723

[27] Sunjae Lee, Hoyoung Kim, Sijung Kim, Sangwook Lee, Hyosu Kim, Jean Young Song, Steven Y. Ko, Sangeun Oh, and Insik Shin. 2022. A-Mash: Providing Single-App Illusion for Multi-App Use through User-Centric UI Mashup. In *Proceedings of the 28th Annual International Conference on Mobile Computing And Networking* (Sydney, NSW, Australia) *(MobiCom '22)*. Association for Computing Machinery, New York, NY, USA, 690–702. https://doi.org/10.1145/3495243.3560522

[28] Sunjae Lee, Hayeon Lee, Hoyoung Kim, Sangmin Lee, Jeong Woon Choi, Yuseung Lee, Seono Lee, Ahyeon Kim, Jean Young Song, Sangeun Oh, Steven Y. Ko, and Insik Shin. 2021. FLUID-XP: Flexible User Interface Distribution for Cross-Platform Experience. In *Proceedings of the 27th Annual International Conference on Mobile Computing and Networking* (New Orleans, Louisiana) *(MobiCom '21)*. Association for Computing Machinery, New York, NY, USA, 762–774. https://doi.org/10.1145/3447993.3483245

[29] Han Liu, Dezhi Han, and Dun Li. 2020. Fabric-iot: A Blockchain-Based Access Control System in IoT. *IEEE Access* 8 (2020), 18207–18218. https://doi.org/10.1109/ACCESS.2020.2968492

[30] Shrirang Mare, Franziska Roesner, and Tadayoshi Kohno. 2020. Smart Devices in Airbnbs: Considering Privacy and Security for both Guests and Hosts. *Proc. Priv. Enhancing Technol.* 2020, 2 (2020), 436–458.

[31] Karola Marky, Alexandra Voit, Alina Stöver, Kai Kunze, Svenja Schröder, and Max Mühlhäuser. 2020. "I Don't Know How to Protect Myself": Understanding Privacy Perceptions Resulting from the Presence of Bystanders in Smart Environments. In *Proceedings of the 11th Nordic Conference on Human-Computer Interaction: Shaping Experiences, Shaping Society* (Tallinn, Estonia) *(NordiCHI '20)*. Association for Computing Machinery, New York, NY, USA, Article 4, 11 pages. https://doi.org/10.1145/3419249.3420164

[32] Meta. 2023. *React Native Learn once, write anywhere.* Meta. Retrieved March 06, 2023 from https://reactnative.dev/

[33] Jakob Nielsen. 1993. *Response Times: The 3 Important Limits.* Retrieved April 4, 2023 from https://www.nngroup.com/articles/response-times-3-important-limits/

[34] Sangeun Oh, Ahyeon Kim, Sunjae Lee, Kilho Lee, Dae R. Jeong, Steven Y. Ko, and Insik Shin. 2019. FLUID: Flexible User Interface Distribution for Ubiquitous Multi-Device Interaction. In *The 25th Annual International Conference on Mobile Computing and Networking* (Los Cabos, Mexico) *(MobiCom '19)*. Association for Computing Machinery, New York, NY, USA, Article 42, 16 pages. https://doi.org/10.1145/3300061.3345443

[35] RAJESH PANDEY. 2023. *How to set up an Amazon Echo for multiple users.* Android Police. Retrieved Mar 3, 2023 from https://www.androidpolice.com/how-to-set-up-amazon-echo-multiple-users/

[36] STEPHEN PERKINS. 2023. *Matter explained: everything you need to know.* Android Police. Retrieved Jan 05, 2023 from https://www.androidpolice.com/matter-smart-home-standard-explained/

[37] Pawani Porambage, An Braeken, Corinna Schmitt, Andrei Gurtov, Mika Ylianttila, and Burkhard Stiller. 2015. Group Key Establishment for Enabling Secure Multicast Communication in Wireless Sensor Networks Deployed for IoT Applications. *IEEE Access* 3 (2015), 1503–1511. https://doi.org/10.1109/ACCESS.2015.2474705

[38] Samsung. 2023. *Invite members using the SmartThings app.* Samsung. Retrieved Jan 17, 2023 from https://www.samsung.com/levant/support/apps-services/invite-members-using-the-smartthings-app/

[39] R.S. Sandhu, E.J. Coyne, H.L. Feinstein, and C.E. Youman. 1996. Role-based access control models. *Computer* 29, 2 (1996), 38–47. https://doi.org/10.1109/2.485845

[40] Amit Kumar Sikder, Leonardo Babun, Z. Berkay Celik, Abbas Acar, Hidayet Aksu, Patrick McDaniel, Engin Kirda, and A. Selcuk Uluagac. 2020. Kratos: Multi-User Multi-Device-Aware Access Control System for the Smart Home. In *Proceedings of the 13th ACM Conference on Security and Privacy in Wireless and Mobile Networks* (Linz, Austria) *(WiSec '20)*. Association for Computing Machinery, New York, NY, USA, 1–12. https://doi.org/10.1145/3395351.3399358

[41] Amit Kumar Sikder, Leonardo Babun, Z. Berkay Celik, Hidayet Aksu, Patrick McDaniel, Engin Kirda, and A. Selcuk Uluagac. 2022. Who's Controlling My Device? Multi-User Multi-Device-Aware Access Control System for Shared Smart Home Environment. *ACM Trans. Internet Things* 3, 4, Article 27 (sep 2022), 39 pages. https://doi.org/10.1145/3543513

[42] Gary Sims. 2023. *How much RAM does your Android phone really need in 2023?* AndroidAuthority. Retrieved Mar 23, 2023 from https://www.androidauthority.com/how-much-ram-do-i-need-phone-3086661/

[43] statista. 2020. *Average number of devices residents have access to in households worldwide in 2020, by country.* Retrieved Mar 3, 2023 from https://www.statista.com/statistics/1107307/average-number-connected-devices-households-worldwide//

[44] Wolfgang Stuerzlinger, Olivier Chapuis, Dusty Phillips, and Nicolas Roussel. 2006. User Interface Façades: Towards Fully Adaptable User Interfaces. In *Proceedings of the 19th Annual ACM Symposium on User Interface Software and Technology* (Montreux, Switzerland) *(UIST '06)*. Association for Computing Machinery, New York, NY, USA, 309–318. https://doi.org/10.1145/1166253.1166301

[45] Shuang Sun, Rong Du, Shudong Chen, and Weiwei Li. 2021. Blockchain-Based IoT Access Control System: Towards Security, Lightweight, and Cross-Domain. *IEEE Access* 9 (2021), 36868–36878. https://doi.org/10.1109/ACCESS.2021.3059863

[46] Madiha Tabassum, Jess Kropczynski, Pamela Wisniewski, and Heather Richter Lipford. 2020. Smart Home Beyond the Home: A Case for Community-Based Access Control. In *Proceedings of the 2020 CHI Conference on Human Factors in*

Computing Systems (Honolulu, HI, USA) *(CHI '20)*. Association for Computing Machinery, New York, NY, USA, 1–12. https://doi.org/10.1145/3313831.3376255

[47] Desney S. Tan, Brian Meyers, and Mary Czerwinski. 2004. WinCuts: Manipulating Arbitrary Window Regions for More Effective Use of Screen Space. In *CHI '04 Extended Abstracts on Human Factors in Computing Systems* (Vienna, Austria) *(CHI EA '04)*. Association for Computing Machinery, New York, NY, USA, 1525–1528. https://doi.org/10.1145/985921.986106

[48] Bergur Thormundsson. 2022. *Adoption rate of major internet and tech services 2022*. Statista. Retrieved Jan 23, 2023 from https://www.statista.com/statistics/1360613/adoption-rate-of-major-iot-tech/

[49] J. Xue, C. Xu, and Yuan Zhang. 2018. Private Blockchain-Based Secure Access Control for Smart Home Systems. *KSII Transactions on Internet and Information Systems* 12 (12 2018), 6057–6078. https://doi.org/10.3837/tiis.2018.12.024

[50] Eric Zeng, Shrirang Mare, and Franziska Roesner. 2017. End User Security &amp; Privacy Concerns with Smart Homes. In *Proceedings of the Thirteenth USENIX Conference on Usable Privacy and Security* (Santa Clara, CA, USA) *(SOUPS '17)*. USENIX Association, USA, 65–80.

[51] Eric Zeng and Franziska Roesner. 2019. Understanding and Improving Security and Privacy in Multi-User Smart Homes: A Design Exploration and In-Home User Study. In *28th USENIX Security Symposium (USENIX Security 19)*. USENIX Association, Santa Clara, CA, 159–176. https://www.usenix.org/conference/usenixsecurity19/presentation/zeng