Lumos: Improving Smart Home IoT Visibility and Interoperability Through Analyzing Mobile Apps

Jeongmin Kim*, Steven Y. Ko†, Sooel Son* and Dongsu Han*
KAIST*, University at Buffalo, The State University of New York, USA†

***Abstract*—The era of Smart Homes and the Internet of Things (IoT) calls for integrating diverse "smart" devices, including sensors, actuators, and home appliances. However, enabling interoperation across heterogeneous IoT devices is a challenging task because vendors use their own control and communication protocols. Prior approaches have attempted to solve this problem by asking for vendor support, or even fundamentally re-designing the architecture of IoT devices. These approaches face limitations as they require disruptive changes.**

This paper explores a new approach to improving IoT interoperability without requiring architectural changes or vendor participation. Focusing on smart-home environments, we propose Lumos that improves interoperability by leveraging Android apps that control IoT devices. Lumos uses this information learned from IoT apps to enable "best-effort" interoperation across heterogeneous devices. Our evaluation with 15 commercial IoT devices from three major IoT platforms and in-depth user studies conducted with 24 participants demonstrate the promising efficacy of Lumos for implementing diverse interoperation scenarios.

I. INTRODUCTION

The smart home market was valued at $76.62 billion in 2018 and is expected to reach $151.38 billion by 2024 [52]. Several major players (Apple, Amazon, Google, and Samsung) are consistently introducing new smart appliances and promoting their own platforms to increase their market share. Along with their popularity, the demand for their *interoperability* has increased [46, 65]. However, it is not trivial to implement a well-integrated system [3, 18, 21, 37, 44, 60].

The difficulties stem from three limitations: 1) Smart home devices often have their own means of control, such as mobile apps, that are proprietary [21, 37]. 2) No single "open" IoT platform is able to cover all IoT devices—SmartThings and Wink, the two largest "open" platforms, respectively support 219 and 115 devices (May 2018), while few devices are supported by both [58, 67]. 3) Interoperation across different IoT platforms is not a design priority.

Existing approaches to IoT interoperation in the smarthome context are based on either voluntary vendor participation [1, 24, 31, 40, 43] or architecture generalization [2, 3, 6, 18, 32, 60, 69]. However, these approaches have their own limitations. The participation-based approaches provide a standardized common interface (e.g., an open API) with which vendors are required to conform. However, due to the extra engineering effort required to support these APIs, these approaches have not been well received; hence a limited number of devices are supported. On the other hand, the generalized architectures propose common data structures and interfaces for heterogeneous IoT devices. Unfortunately, they have not been widely deployed because they demand significant changes to the current IoT architecture and require the agreement of all stakeholders.

We posit that interoperability will remain a long-lasting problem in the foreseeable future and seek for an alternative solution that can be driven by users. Thus, we explore a different approach, in which we leverage only the information already available, enabling interoperation of IoT devices without explicit vendor support or architectural changes. From this perspective, the core challenges are to: 1) obtain *visibility* and *controllability* for IoT devices only from the available information; and 2) improve interoperability among existing IoT architectures. To this end, we design a new system called Lumos that empowers users with an automated framework that supports interoperability. Lumos leverages the key insight that many IoT devices for smart homes are controlled by Android apps [26, 27, 53]. The apps are readily available and already know how to control IoT devices and query device information.

Leveraging information obtained through app analysis, Lumos improves the interoperability with minimal user configuration. To infer the semantics and control/status messages generated by an app, Lumos combines UI, program and traffic analyses performed on the app binary. It then integrates the analysis results with finer-grained semantic information from the user who best knows the context of the action she triggers when using the app. Based on the information, Lumos creates an interoperation gateway that understands the semantics of messages between the app and its IoT device and actively sends control messages and status queries to the IoT device. Finally, Lumos helps users easily create interoperation scenarios of their own by automatically generating scripts. Our contribution shows that a pure user-driven approach is viable in bridging the gap until the market fully resolves the problem.

We evaluate Lumos using 15 commercial off-the-shelf smart-home devices. We show that Lumos can learn the semantics of IoT operation from network traffic for all device features (29 out of 29), and is able to generate status and control messages for most features (26 out of 29, §V-B). Finally, our user study with 24 participants shows that Lumos offers a practical programming framework that enables the interoperation of IoT devices, offers interoperation features that are not available on commodity IoT platforms, and requires reasonable configuration effort compared with three

978-1-7281-6992-7/20/$31.00 ©2020 IEEE

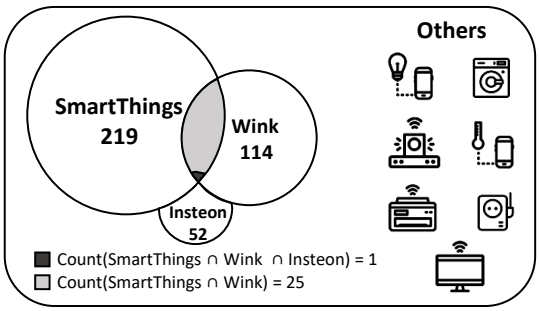


Fig. 1: Fragmented smart home ecosystems

popular IoT platform-native apps.

In summary, this paper makes two key contributions:

- **Novel approach to interoperability:** We present an automated framework that combines static program analysis and dynamic learning to understand and re-construct control messages with user-given semantics.
- **System prototype and evaluation:** Our in-depth evaluation shows how Lumos enables interoperation between IoT devices in a unilateral fashion, further enabling new value-added home IoT services. To the best of our knowledge, our evaluation covers the *largest* set of commercial IoT devices among existing work. Our user study with 24 participants shows the promising efficacy of Lumos and quantifies the user effort.

II. Motivation

A. The Status Quo of IoT Interoperation

Market reports [29] indicate the existence of 450 IoT platforms world-wide as of 2017, marking a 25% increase since 2016 and showing how the IoT ecosystem is becoming increasingly fragmented. *Interoperability* is the ability to create a coherent service by interacting with multiple IoT devices. Many studies confirm that such fragmentation presents a significant barrier that impedes interoperability and wider adoption of home IoT services [19, 28, 30, 44, 68]. We often come across users experiencing frustrations and ordeals as they try to make different IoT devices interoperable [51, 54].

Fig. 1 shows three major IoT platforms and the number of devices they support. We make the following observations:

- Only devices on the same platform are interoperable. Cross-platform inter-operation is generally not supported. Out of the three, only SmartThings exposes external APIs for controlling and monitoring devices [59].
- If devices belong to a platform, they are locked-in to that specific platform. They neither support multiple platforms nor change their platform. We suspect that this limitation originates from implementation costs and business partnerships [57, 61].
- Many IoT devices are still stand-alone and do not interoperate with other devices. For example, Chromecast [23] does not belong to any of the three major platforms and cannot interact with any devices on these platforms.

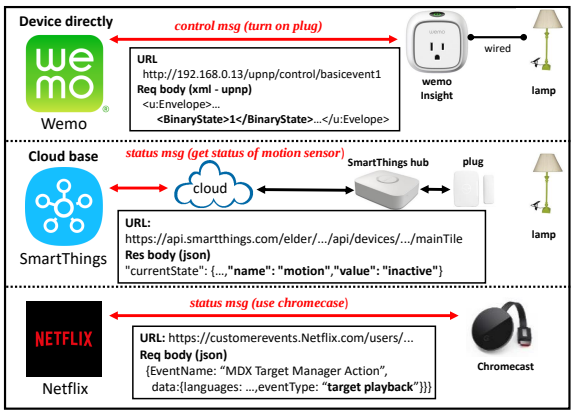


Fig. 2: Connection types of devices and network traffic

Industry efforts: Notable approaches to this problem from the industry include OpenT2T [43] and IFTTT [24]. OpenT2T defines common IoT schemas, which consist of properties for similar devices. For example, a common schema for IoT light bulbs can define an "on-off" property with two possible values, "on" and "off". These schemas are then translated into vendor-specific implementations. A common schema then provides a consistent user experience when operating similar devices, even when they are from different manufacturers or support different protocols. While the idea of abstraction is noble, it requires vendor participation to support common schemas. No tangible incentive for this participation has contributed to rendering OpenT2T inactive for more than three years at the time of writing [42].

IFTTT [24] allows users to write Applets that connect popular web services, apps, and IoT devices. A user can set up triggers that specify when an Applet should run, filters that express a desired condition, and actions that are executed when filter conditions are met. Combined with IoT devices, it can be used to customize IoT services. However, it leverages existing open APIs to access device state and issue commands, which requires vendor supports. Even when vendors expose open APIs, we find that they expose only a small set of features.

B. Challenges and Key Insight

Approaches that require vendor participation or architectural change are still far from wide-deployment, despite the growing needs for interoperability among users. It is necessary to fully understand IoT devices (e.g., protocol) even when we aim to improve interoperability without vendor participation. However, it is difficult to obtain such understandings since almost all vendors do not open this information (§II-A). Furthermore, they do not wish to incur the high implementation costs that design changes entail [50]. The main challenges are to 1) capture the device state for context monitoring (*visibility*) and issue a desired command (*controllability*) without vendor support and 2) improve interoperability without requiring architectural modifications. We take the following usecase as a concrete running example to illustrate the problem. Alice has an IoT light bulb and a streaming dongle (e.g., Chromecast [23]). Each device has a companion mobile app that allows Alice to control the device. Now, Alice wants to

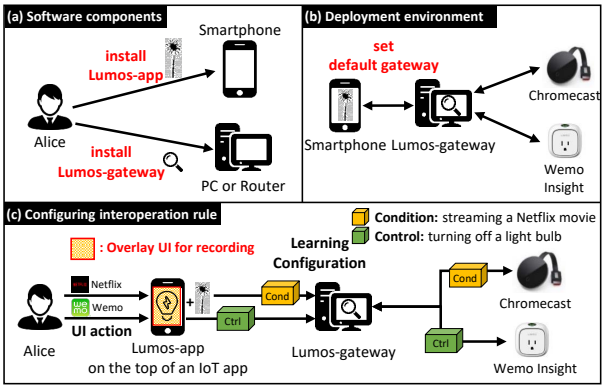


Fig. 3: Usage Model - System components, Deployment environment, and Configuring interoperation rule

configure them together so that the light bulb can automatically turn itself off when she streams a movie to her TV. A smarthome system should be aware of whether the streaming device is currently playing a video and be able to issue the control command to turn off the light bulb. Once the two tasks are addressed, implementing an IoT service with multiple devices amounts to writing a composition rule.

Key insight: Our goal is to enable the two tasks in a user-driven, best-effort manner with automation support. Our key insight is that mobile apps play a key role in communicating with IoT devices and contain valuable information for interoperability. 1) They already have the ability to control and monitor IoT devices. 2) Often vendors themselves provide the apps and keep them up-to-date. 3) The graphical user interface (GUI) of the apps provides semantic information (e.g., this button turns off the light).

Fig. 2 presents an example in which Alice uses a Wemo Insight Plug to control room lighting. When Alice watches a Netflix movie using Chromecast, the Netflix app sends an HTTP request message to the server. This request contains a message ("eventType":"target playback") denoting that the movie is to be played on the TV connected with Chromecast. When the request is detected, Lumos triggers a request to the Wemo Insight Plug to power off. This is feasible because Lumos learns from IoT apps to *recognize* the condition and *generate* the control message.

III. Lumos Usage Model

In order to use Lumos, a user installs two software components (Fig. 3 (a)). One component is Lumos-app, a mobile app that allows users to configure interoperation. The other component is Lumos-gateway, a middlebox that can be installed on either a desktop or a router that can run a custom OS (e.g., a NETGEAR router). Lumos-gateway is a trusted party that monitors all traffic that IoT apps generate (Fig. 3 (b)). To monitor the traffic, users need to configure their devices to use Lumos-gateway as the default gateway. Note, an IoT app may directly connect to an IoT device (e.g., Wemo Insight Plug) or go through an IoT hub that talks to IoT devices (Fig. 2). Lumos-gateway supports both cases.

Configuring interoperation: Our interoperation rule consists of a condition and a control action. For example, in our running example with Alice, the condition is streaming a Netflix movie and the control action is turning off the light. When Lumos detects the condition, it performs the control action. To configure a rule, users "teach" Lumos by performing UI actions that correspond to the condition and the control. For example, Alice teaches Lumos her condition (streaming a Netflix movie) by opening her Netflix app and playing a movie. She also teaches her control (turning off a light bulb) by opening her Wemo app and turning the light bulb off.

We automate this process by capturing the user interaction with Lumos-app that runs in the background and displays an additional UI overlaid on top of an IoT app UI. The additional UI displayed by Lumos-app guides the user through the process of configuring a condition action and a control action. While a user configures a condition action and a control action, the Lumos-app monitors the UI actions performed by the user and communicates with Lumos-gateway to capture the requests and responses caused by these UI actions at the network level, as illustrated in Fig. 3 (c). Then, Lumos-gateway analyzes the requests and responses to detect the condition and trigger control actions. Our implementation of Lumos-app extends an existing UI record-and-replay tool called SUGILITE [36], which relies on the Android Accessibility API to monitor, intercept, and inject UI actions.

IV. Design

Achieving our goal requires satisfying three requirements: 1) Lumos-app must provide a way for users to leverage IoT app UIs to "teach" our system of their intended conditions and control actions. 2) Lumos-gateway must be able to recognize pre-configured conditions from the network messages that IoT apps generate and issue control messages to trigger desired actions. 3) It must offer programmability using the "learned" information to configure interoperation rules. Fig. 4 presents a system overview that delivers the goal. We detail each component of our design below.

A. Learning from UIs and Users

Lumos starts by learning the semantics of IoT operation through the UI and user actions on an app. The objective of this process is to identify the actions of interest that generate control/status messages and label them with a specific semantic tag. This tag denotes an IoT device operation controlled via an IoT app UI component, such as turning on/off a bulb. Lumos takes a user-assisted semantic labeling approach to reduce human effort. In the 'teaching' phase, when a user clicks a UI component, Lumos assigns the resource ID of the UI component as the semantic tag because a resource ID is a human-readable string that usually has semantic information (e.g. brightness_slider). This is done by Lumos-app by monitoring user interactions through the Android Accessibility API. However, the tag might be insufficient. Some resource IDs do not contain any semantic information (e.g., button1), or a single button may trigger different actions depending on the context (e.g., a single switch button for power on and off).

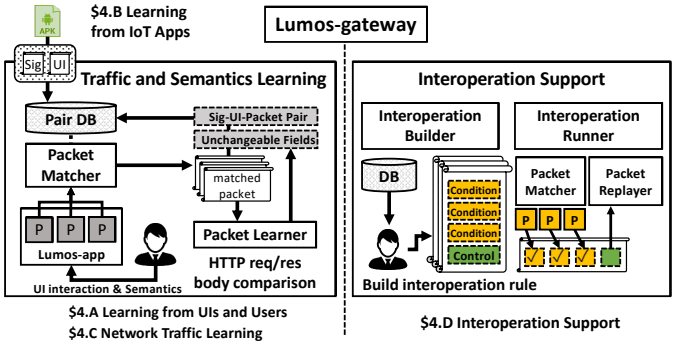


Fig. 4: Lumos system overview

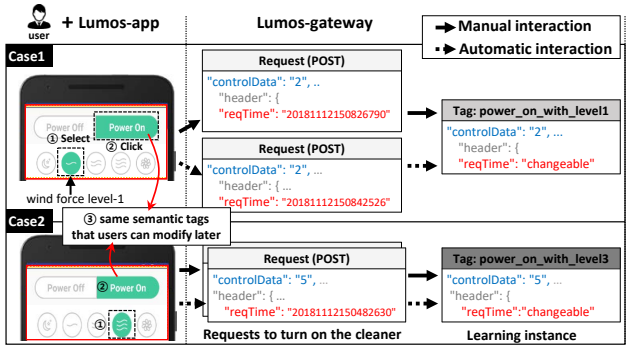


Fig. 5: Dynamic learning examples of WINIX app

To manage this, Lumos allows a user to edit semantic tags displayed during the 'teaching' phase to make the meaning more specific and personalized to the user. For example, when Alice wants to teach the system how to turn on a WINIX air cleaner using its app, she demonstrates it by clicking the power-on button while Lumos-app is running. Lumos-app then records all of the interactions. When the power-on button is clicked, Lumos-app shows a dialog with the default semantic tag ("power on"), which the user can then edit for customization.

Example: Fig. 5 (left) illustrates the teaching phase in which a user teaches Lumos-app how to turn on a WINIX air cleaner with a specific wind force level. ① She sets the wind force options. ② She then turns on the cleaner. ③ Lumos-app automatically assigns the resource ID ("power on") to the power-on button as the semantic tag. Lumos-app records these interactions for a later step of automatically replaying these operations (§IV-C) and enables to customize the tag with a concrete meaning (e.g. power on with wind force level 1).

B. Learning from IoT Apps

By design, Lumos-gateway should trigger a programmed action upon observing network messages that represent an IoT operation. This architecture necessitates Lumos-gateway to identify HTTP(S) requests that an IoT app generates when clicking a particular UI component. Unfortunately, identifying such requests is non-trivial without an understanding of the app logic due to other unrelated requests in the background.

To illustrate this, we quantify the amount of traffic generated while performing specific actions on the IoT apps listed in Table I. We capture traffic from each app from the time the target app is started, perform UI interactions as quickly as possible, and stop capturing traffic immediately. We report the average of 10 runs. We perform nine actions using the apps: 1) lock an August door lock; 2) stream to a Chromecast; 3) turn on a HUE bulb; 4) power on an Insteon plug 5) get status from Nest Protect; 6) power on a SmartThings plug; 7) power on a Wemo Insight; 8) play Wink chime; and 9) power on a Winix air cleaner. On average, each app generates 27.4 HTTP transactions. According to our manual traffic analysis, all apps in our dataset continuously perform synchronization as long as the app is in the foreground. We suspect this is due to the need for minimizing synchronization delays to enrich user experience. To isolate transactions that a UI component generates, Lumos uses static program analysis. It takes an Android binary (APK) and then pairs a UI component with the regex signatures of control/status messages that the UI component generates. In addition, it tracks dependencies between messages to dynamically learn fields that come from previous messages.

Building network signatures: To identify the exact control and status message that a UI component generates or displays after receiving it, we start from an existing tool, Extractocol [33], that conducts a static taint analysis to extract network message signatures that an Android app generates or receives. It automatically identifies all app-defined methods that send network messages, extracts message signatures, and outputs them in regular expressions. Fig. 6 shows a regular expression example. Extractocol also provides a call graph of an app as well as its control-flow and interprocedural data-flow graphs. We leverage the information in the next phase of our analysis.

UI control identification: Given an app's call graph and its network signatures, we associate each message signature with a UI component that generates a network message matching the signature. The goal is to precisely identify the network messages generated by the UI actions of a user when the user configures conditions and control actions. The Android accessibility API allows the monitoring of which UI components a user interacts with by providing their resource IDs. If we can associate a resource ID with a network message signature, we can isolate the traffic that the UI component generates from Lumos-gateway, which observes the traffic.

Lumos takes the following two steps to accomplish the goal. First, it identifies all event listeners that eventually generate network messages. Since every interactable UI component has an event listener, identifying an event listener is equivalent to identifying a UI component. Lumos does a backward call graph analysis for this; it starts from each app-defined method that generates a network message until it either finds an event listener such as `OnClickListener()` and `OnSeekBarChangeListener()`, or reaches the top of the call chain. Second, Lumos identifies the resource ID to which an event listener is attached. Android allows a developer to register an event listener of a UI component either dynamically (in the app code) or statically (in the app's XML manifest). Statically-registered event listeners are not difficult to identify since it simply requires parsing of

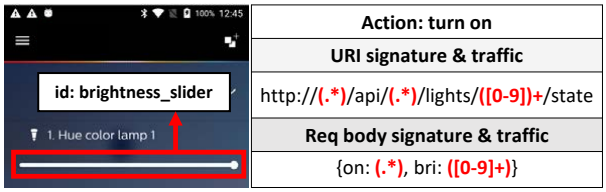


Fig. 6: Examples of Network signature, UI control, and string ID for HUE app

the XML manifest. However, dynamically-registered event listeners are not straightforward to identify. Thus, Lumos performs further analysis to identify dynamically-registered event listeners. Lumos starts this analysis by looking up the call site of every event listener registration method (e.g., `setOnClickListener()`). From there, it finds all objects that invoke the event listener registration methods. These objects are user-interactable UI objects. Subsequently, Lumos performs backward taint analysis for each user-interactable UI object until it finds a method that provides the resource ID for the object, such as `findByViewID()`. This resource ID is a hexadecimal, and we can find the corresponding string ID in another XML file (`public.xml`).

Example: Fig. 7 shows how Lumos couples a UI component and the control message it generates using the HUE app. Extractocol outputs a request sending method, which Lumos identifies to be (eventually) by `OnSeekBarChangeListener`. Therefore, Lumos looks for a `setOnSeekBarChangeListener` call site that registers the event handler. From the call site, it computes a backward slice that affects the object (`this.a`) to which the event handler was attached. It finally reaches the UI's resource ID ("0x7F0D009C"), which is labeled as a "brightness_slider" in `public.xml`.

C. Learning from Network Traffic

The UI-signature pairs we extract from an app allow us to distinguish the traffic triggered by the app's UI from other traffic. However, statically-extracted message signatures do not provide run-time values, such as URIs, query strings, and headers. Yet, Lumos should be able to construct a network request for monitoring and controlling the status of a device. This means Lumos must know how to fill in actual values.

To address this, Lumos integrates a run-time packet learning module with information learned from a static analysis. For this, Lumos-app replays all of the interactions recorded in the learning phase of §IV-A to generate network traffic. Lumos-gateway then detects and captures the network messages that match the network signatures extracted from the phase in §IV-B. With this task, Lumos-gateway obtains multiple message instances for each network signature, as shown in Fig. 5, and uses the values obtained from the observed network messages. However, some of the attribute values change over time (e.g., message timestamp), which means that Lumos-gateway should identify attributes that change across message instances and should not perform exact matching on those attribute values when recognizing a pre-configured condition.

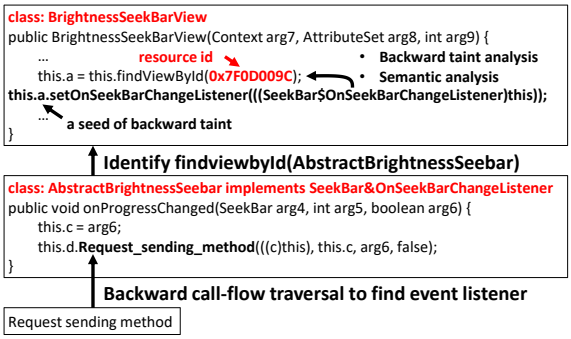


Fig. 7: HUE app UI finding example

On the other hand, to generate valid control messages, Lumos must accurately reconstruct them on the fly.

To this end, for each message attribute that changes dynamically, Lumos computes the program slice that is responsible for generating this attribute from a target IoT app via leveraging the program analysis module. Lumos computes this slice in the following steps: 1) The analysis module identifies a seed statement that inserts the attribute into the message; 2) It performs a backward taint analysis of computing statements that have transitive data dependencies on the seed statement.

Automated re-learning: The communication between IoT devices and apps often uses REST APIs with an authorization token. Because the communication happens (mostly) over HTTPS and the device assumes that a token holder is a trusted party, tokens are often reused. However, a token may be updated and old ones may become invalidated, in which case Lumos-gateway has to learn a new token. Lumos uses two methods to automate this. It learns a new authorization token by monitoring network traffic and leveraging the dependency relationship between network messages analyzed by the static analysis module (as described in §IV-B). For example, when a user operates a door lock by using the August app, the app sends a request that includes the token from the previous response's header ("x-august-access-token"). If the request is valid, the server sends a corresponding response with a new token, which gets reused in subsequent requests. Lumos-gateway automatically tracks the refresh between the app and server. In the second method, Lumos actively generates messages that include a new token by leveraging the app. In this method, Lumos-gateway sends a push message to Lumos-app for token re-creation. If Lumos-app receives the message, it replays a UI interaction recorded from the previous learning phase which creates a new token. Lumos-gateway then monitors generated network packets and acquires the token from the relevant packets. This method is used when an active monitoring or action fails due to token expiration.

D. Interoperation Support

Our rule builder allows users to configure desired interoperation rules that consist of multiple conditions and control actions, enabling complex interoperation scenarios across heterogeneous IoT devices. The interoperation run-time monitors the conditions. When a condition is satisfied, it performs the corresponding control action.

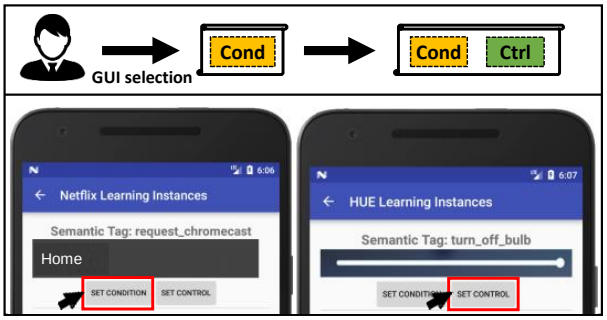


Fig. 8: Rule specification on Lumos-app

Rule builder: Lumos allows users to compose interoperation rules with conditions and control actions. We support two types of conditions: 1) a "passive" condition is detected by monitoring network traffic without Lumos-gateway sending any traffic, and 2) an "active" condition involves Lumos-gateway asking for a device status directly by sending a learned network message (e.g., motion sensor is active or not). When an active condition is set, Lumos-gateway periodically checks the device status without a user's direct action.

Fig. 8 shows how Alice configures turning off her bulb in the living room when playing a Netflix movie on a TV connected to Chromecast. She starts Lumos-app and selects the configuration menu. The app, then, shows a list including UI components with semantic tags (e.g., "turn_off_bulb"). Alice chooses a desired semantic tag ("request_chromecast") in the list, and sets the tag as a passive condition as shown in Fig. 8 (a). As Fig. 8 (b) shows, she then selects the control to turn the bulb off. This concludes Alice's configuration steps. Lumos-app then registers the rule to Lumos-gateway. Our demo video shows this scenario with Lumos-app: https://youtu.be/rdrDUL7-9Zs.

Interoperation run-time: Once a rule is registered, Lumos-gateway monitors network traffic to check for the condition of the rule and performs the desired action when the condition is satisfied. For the condition detection, Lumos-gateway uses exact matching for the unchangeable fields in the condition's HTTP request body. Once the rule is registered, the run-time checks whether a network message matches the condition of the rule. When it matches, the run-time generates an HTTP request that triggers the corresponding action with an up-to-date authorization token.

E. Practical Issues

Information reuse: The network signature-UI pairing is a one-time setup procedure. To facilitate the adoption, Lumos supports the import and export of network signature-UI pairs. Note this setup file does not contain any credentials (e.g., authorization token, ID, and password) because it only contains signature-UI pairs extracted from IoT apps, which are public information that anyone can extract. After importing this setup file, users only need to perform UI interactions for interoperation rule configuration. We implemented this feature and measured the time to analyze the apps and load the information. For the nine IoT apps we use in our evaluation, Lumos took a total of 36.5 minutes for analysis. However, loading the result took only 0.95 seconds for the nine apps. This shows the effort of initial learning can be amortized across users.

Handling SSL traffic: To support Android apps using SSL for encrypted communications with their server-side services, Lumos-gateway deploys a local certificate authority (CA) certificate, which is used prevalently in the industry [20, 47, 48, 62]. Specifically, a user installs a Lumos issued certificate into an IoT app, which allows Lumos-gateway to inspect traffics between the app and its server-side service. Lumos-gateway then re-encrypts the inspected traffics using a session key derived from the original certificate, thus preventing man-in-the-middle entities.We applied this interim solution to eight IoT apps that we evaluated due to their SSL usage.

System overhead: We analyze battery consumption and computation latency in Lumos-gateway. First, an "active" condition, the interval of which can be configured (2s by default), periodically checks the device status, which may result in battery drains. However, queries for active conditions are processed by the hub, not the end devices. For example, Wink and SmartThings hubs cache the latest status of connected devices [55, 66]. None of the devices in our evaluation set consume device battery when Lumos-gateway sends active queries. In our evaluation setup, Lumos-gateway runs on a PC with Intel i5 and 16 GB memory, and the average packet processing time (monitoring and sending messages) in Lumos-gateway is 143ms, which is not huge overhead in terms of usability. However, it is possible that the latency would increase with commodity home routers as they have less computing power than a typical PC. Thus, it requires further engineering effort to optimize Lumos-gateway's packet processing for real-world deployment with commodity routers.

V. Evaluation

We evaluate Lumos by answering three key questions:

- *Does Lumos enable interoperation across diverse IoT devices and platforms?* **(RQ1)**
- *Is Lumos-gateway capable of emulating key functionalities of the IoT apps?* **(RQ2)**
- *How easy is it for a user to configure interoperation rules using Lumos?* **(RQ3)**

We evaluate Lumos using 15 commercial IoT devices and nine apps that control them, as listed in Table I. To show the feasibility of Lumos, we use standalone IoT devices and devices from three popular home IoT platforms. Five devices are standalone: an August smart lock, Chromecast, a HUE bulb, Nest Protect, and a Wemo Insight. The rest belongs to one of the three popular platforms: SmartThings, Wink, and Insteon.

A. Supporting Diverse IoT Platforms (RQ1)

We first evaluate whether Lumos can emulate the key functionality of IoT apps to monitor and control the devices.

Devices			Supported Platform			Lumos
Application	Device	Type	Insteon(3/15)	SmartThings(5/15)	Wink(6/15)	(15/15)
August	August smart lock pro	Door lock	n/a	✓	n/a	✓
Netflix	Chromecast	Streaming-dongle	n/a	n/a	n/a	✓
Philips HUE	HUE	Bulb	n/a	✓	✓	✓
Insteon	Insteon door sensor	Door sensor	✓	n/a	n/a	✓
	Insteon plug	Smart plug	✓	n/a	n/a	✓
	Insteon water leak sensor	Water leak sensor	✓	n/a	n/a	✓
Nest	Nest Protect	CO&smoke detector	n/a	n/a	✓	✓
SmartThings	SmartThings plug	Smart plug	n/a	✓	n/a	✓
	SmartThings motion sensor	Motion sensor	n/a	✓	✓	✓
	SmartThings door sensor	Door sensor	n/a	✓	n/a	✓
Wemo	Wemo Insight Plug	Smart plug	n/a	n/a	n/a	✓
Wink	Wink chime	Siren&chime	n/a	n/a	✓	✓
	Wink door sensor	Door sensor	n/a	n/a	✓	✓
	Wink motion sensor	Motion sensor	n/a	n/a	✓	✓
Winix	Winix air cleaner	Air cleaner	n/a	n/a	n/a	✓

TABLE I: IoT devices, platforms and interoperability (Insteon, SmartThings, Wink and Lumos).

Table I summarizes the coverage of Lumos compared to the three popular IoT platforms. The "device" column family lists 15 IoT devices and their corresponding mobile apps. *Platform-native* apps (shaded) provides programmable features for various devices within the platform. In contrast, *stand-alone* apps do not provide any interoperation features. The "supported platform" column shows how many devices the three popular IoT platforms support. In our evaluation benchmarks, SmartThings and Wink support only five and six devices, respectively. Insteon platform devices cannot interoperate with any of the other two platforms, clearly demonstrating their mutually exclusive nature. In contrast, Lumos provides much wider coverage than existing platforms.

B. Emulating IoT App Functionalities (RQ2)

Our next question is whether or not Lumos is able to emulate key functionalities of the IoT devices for each feature they support. Table II shows the coverage results compared to OpenT2T. The "OpenT2T" column shows whether the feature is supported by OpenT2T, and the "Lumos" column indicates whether Lumos is able to perform the same operation.

Using real-world IoT devices, we examine whether each individual control and status messages is recognized and Lumos-gateway's control action triggers the correct device behavior. App features that involve device status queries (shaded in the table) can be used as a passive condition ('PC') and/or an active condition ('AC') in Lumos (§IV-D). Control functions that trigger an action on a device can be used as passive conditions ('PC') and/or control actions ('C'). Note, control actions and active conditions require Lumos-gateway to generate actual messages and succeed in achieving the desired outcome. To support passive conditions ('PC'), Lumos-gateway must be able to recognize the network messages.

Lumos supports all 18 status query features as either passive or active conditions and all 11 control features as either passive conditions or control actions. In fact, with one exception (Chromecast), it supports both.

In contrast, seven out of 15 devices (e.g., door locks, streaming dongles, etc.). do not have an OpenT2T schema. To enable interoperability using OpenT2T, vendors need to have a pre-defined schema for a new device. Out of the 29 features that 15 devices support, only 10 are supported by OpenT2T. Even the devices that are supported by OpenT2T do not expose all their features (e.g., Nest Protect). This is because either the schema only defines minimal features for each device type or the vendor partially implements the translation from the common schema to its own features. The large difference between Lumos and OpenT2T (29 versus 10 features supported) shows the benefit of our approach, which is not tied to any common interface that must be agreed upon, and does not require vendor support. We observe Lumos does not support control action for the Netflix-Chromecast case when the Netflix app goes through a server to talk to a Chromecast device and their messages do not carry context information (e.g., what movie is being played). This is the only case we have observed where the API violates statelessness. Thus, Lumos does not trigger any action on Chromecast.

C. User study (RQ3)

To answer our usability question, we conducted a user study in which we asked our participants to configure Lumos with new interoperations for five given missions.

Participants and environment: Twenty-four participants were recruited, all of whom were all university students. They were required to be active smartphone users of age 20 or older. We asked them to mark their own IoT experience on a three-point scale: 1) "no experience"; 2) "having experience with at least one IoT device"; and 3) "having programming experience on at least one IoT platform". The groups for each score on the scale consisted of nine, eight, and seven people, respectively. The experiment was conducted in a test room with 15 IoT devices and three IoT platforms, as listed in Table I. We installed Lumos-gateway which runs on a PC with Intel i5 and

#	Device	App function	OpenT2T	Lumos	#	Device	App function	OpenT2T	Lumos
1	August smart lock pro	lock/unlock	No schema	PC, C	9	SmartThings motion sensor	active or not	✓	PC, AC
		get status history		PC, AC			get status history	Not supported	PC, AC
2	Chromecast (Netflix)	request Chromecast	No schema	PC	10	SmartThings door sensor	open or not	No schema	PC, AC
							get status history		PC, AC
3	HUE	turn on/off	✓	PC, C	11	Wemo Insight Plug	power on/off	✓	PC, C
		change brightness	✓	PC, C					
		change color	✓	PC, C			get current voltage	Not supported	PC, AC
		get status	Not supported	PC, AC					
4	Insteon door sensor	open/close status	No schema	PC, AC	12	Wink chime	play bell	No schema	PC, C
5	Insteon plug	power on/off	✓	PC, C	13	Wink door sensor	open or not	No schema	PC, AC
							get status history		PC, AC
6	Insteon water leak sensor	get leak status	✓	PC, AC	14	Wink motion sensor	active or not	✓	PC, AC
							get status history	Not supported	PC, AC
7	Nest Protect	get CO status	✓	PC, AC	15	Winix air cleaner	turn on/off	No schema	PC,C
		get smoke status	Not supported	PC, AC			change wind force		PC, C
		get battery health	Not supported	PC, AC			get current status		PC, AC
8	SmartThings plug	power on/off	✓	PC, C			–		
		get status history	Not supported	PC, AC					

TABLE II: Functionality comparison (App/Lumos/OpenT2T schemas). 'C' stands for control, 'PC' for passive condition, and 'AC' for active condition. Features in grey represent status query functions. The rest are control functions.

Mission	# devices	Clicks$_{Teach}$	Clicks$_{Conf}$	Condition
Tutorial$_{\{1,1\}}$	2	5	5	-
Insteon$_{\{1,1\}}$	2	6	6	-
Stand-alone$_{\{1,2\}}$	3	6	6	-
SmartThings$_{\{2,2\}}$	4	9	10	AND
Wink$_{\{2,2\}}$	3	10	9	OR

TABLE III: Mission overview: "Clicks$_{Teach}$" stands for the minimum number of clicks required for teaching, "Clicks$_{Conf}$" stands for the minimum number of clicks required for configuration. "Condition" stands for the relationship among multiple conditions.

16G RAM. We provided the participants with a smartphone with the nine apps and Lumos-app.

Missions: We assigned each user five missions that are based on common interoperation scenarios. To directly compare Lumos-app with apps provided by the existing IoT platforms that provide programmable interfaces for automation, we designed three missions to program an interoperation rule using the existing platforms as well as Lumos. The first mission was used as a tutorial in which the experimenter showed a participant how to teach Lumos and how to configure interoperations with our system. The other missions were given to the participants in a random order. After the tutorial, we gave users some time to familiarize themselves with the apps (August, Netflix, Insteon, SmartThings, Wemo, HUE, and Wink). During each mission, we did not answer questions on how to use Lumos-app. The missions are as follows.

Tutorial$_{\{1,1\}}$*:* The mission assumes that a user wants to turn off a bulb connected to a Wemo Insight when streaming a Netflix movie through Chromecast. In this mission, we demonstrate the procedure of configuring interoperation. The participants need to teach Lumos-app using Netflix and Wemo apps and then configure interoperation that is made up of *one condition* and *one control* (hence $\{1,1\}$). The standard procedure for this mission has five clicks for teaching ("Clicks$_{Teach}$") and five clicks for configuration ("Clicks$_{Conf}$").

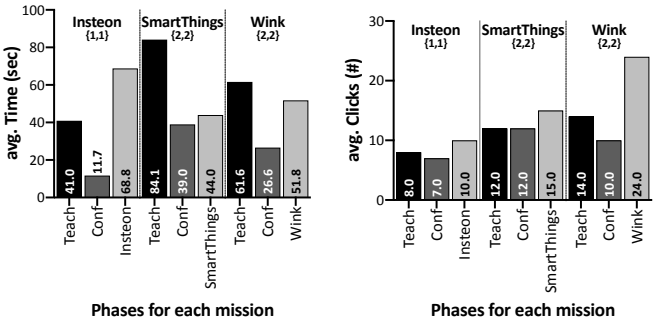
Insteon$_{\{1,1\}}$*:* This mission requires participants to configure interoperation to turn off an Insteon plug when an Insteon water leak sensor is triggered. This mission is relatively easy because participants use the Insteon app only to control those devices. This interoperation consists of *one condition* and *one control* with six "Clicks$_{Teach}$" and six "Clicks$_{Conf}$".

Stand-alone$_{\{1,2\}}$*:* This mission requires turning off a Winix air cleaner and a Wemo Insight when an August door lock is locking, which consists of *one condition* and *two controls*. Participants need to teach Lumos-app using three IoT apps, as listed in Table III. The standard procedure for this mission has eight "Clicks$_{Teach}$" and six "Clicks$_{Conf}$". Note only Lumos supports this scenario unlike other existing IoT platforms.

SmartThings$_{\{2,2\}}$*:* This mission assumes that a user turns off a SmartThings plug and a HUE bulb connected to a SmartThings hub if a SmartThings motion sensor does not detect any motion and a door sensor is closed. Thus, the interoperation for this mission consists of *two conditions connected by "AND"* and *two controls*. To configure the interoperation, participants teach Lumos-app by using the SmartThings app to control four devices (motion sensor, door sensor, plug, and bulb). The standard procedure has nine "Clicks$_{Teach}$" and eight "Clicks$_{Conf}$". Besides, participants need to select the condition ("AND").

Wink$_{\{2,2\}}$*:* This mission requires participants to configure interoperation to play a Wink siren when a Wink door sensor is opened, or a Wink motion sensor detects motion. Therefore, the interoperation consists of *two conditions by connecting "OR"* and *one control*, which requires 12 "Clicks$_{Teach}$" and nine "Clicks$_{Conf}$". Like SmartThings$_{\{2,2\}}$, participants need to choose an appropriate condition ("OR").

Procedure: The experiment took about one hour per participant. After signing the IRB-approved consent form, each participant performed a full rehearsal of the tutorial with the



(a) The avg. completion time (b) The avg. completion click

Fig. 9: The comparison of three platform-native apps and lumos-app

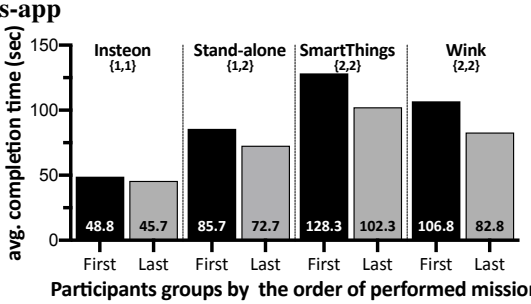


Fig. 10: The average completion time (sec) grouped by the order of start mission. "First" stands for a group that conducted a specific mission first. "Last" stands for a group that performed a specific mission last.

experimenter. They also learned how to configure automation with three platform-native apps during that time. Following the tutorial, the experimenter gave four missions in random order. For a given mission, a participant first performed it using Lumos-app. For the three missions with existing IoT-platform support (Insteon$_{\{1,1\}}$,SmartThings$_{\{2,2\}}$, and Wink$_{\{2,2\}}$), the participants performed them again using the platform-native apps. Note that, there are some nuanced differences in usability across platforms, which we will discuss later in the section.

For each mission, we measured the number of clicks and time required for mission completion for the teaching step and the configuration step separately. In the teaching step, Lumos-app automatically records the count of click operations from the "start recording" click to the "end recording" click. Likewise, in the configuration step, Lumos-app records the count from the "interoperation manager" click to the "finish configuration" click.

Mission results: Overall, 23 out of the 24 (95.8%) participants succeeded in all four missions and one succeeded in three missions. The participant who failed a mission did not complete SmartThings$_{\{2,2\}}$ because the participant prematurely clicked "configuration finish" button from confirmation dialog before completing configuration; the participant was supposed to select the next condition and the 'AND' operator to combine the two conditions. Fig. 9 compares Lumos-app and three platform-native apps, in terms of mission completion times and clicks.

How does Lumos-app compare to platform-native apps in terms of time and effort to configure interoperation? To demonstrate that Lumos-app has reasonable entry barriers, we compared platform-native apps and Lumos-app for Insteon$_{\{1,1\}}$, SmartThings$_{\{2,2\}}$, and Wink$_{\{2,2\}}$ (Fig. 9). In general, using Lumos-app for all missions except Insteon$_{\{1,1\}}$ took more time, and an equal or more number of clicks compared to existing platform-native apps. This is because unlike platform-native apps, Lumos involves teaching which takes most of the user time. When looking at time and clicks for configuration alone excluding teaching, Lumos-app is considerably faster. We note that the teaching results are reusable. If Lumos-app has already finished learning a specific IoT device's action, users just need to decide whether it is a condition or a control in the configuration phase. The teaching phase can even be omitted using information sharing and automated re-learning, thereby reducing the time and effort considerably (§IV-E).

How does Lumos-app compare to platform-native apps in terms of usability? We found nuanced differences in the programmability of the three IoT platforms. Unlike Lumos, the platform-native apps do not easily support relationships to connect thenmultiple conditions that are required by our missions, Wink$_{\{2,2\}}$ and SmartThings$_{\{2,2\}}$. SmartThings supports this only through the script-based programming API to complement their usability [56]. Wink and Insteon do not provide ways to combine multiple conditions. To work around this, we prepared a SmartThings app script (about 80 lines) in advance for SmartThings$_{\{2,2\}}$ and uploaded it to the SmartThings marketplace from which the participants downloaded and chose a proper device required by the script.

For Wink$_{\{2,2\}}$ that uses two conditions connected by the "OR" condition, participants had to resort to breaking down the rule into two separate rules. This is the reason Wink$_{\{2,2\}}$ required more clicks than other missions. Finally, the Insteon$_{\{1,1\}}$ mission required the most time when using the Insteon platform app (Fig. 9(a)). This is because Insteon takes some time (about 30s) to apply automation rules to the desired devices [25] For this, six participants left negative feedback that the process was boring and inefficient.

Unlike Insteon, Lumos instantly configures the rules without any noticeable inefficiency. Although we concede that the usability of platform-native apps can improve, our comparison to the current status quo shows that Lumos offers a more efficient programming environment and comparable levels of effort and difficulty as platform-native apps.

Learning curve for users. We look at whether the time to complete missions decreased as users gained more experience. Recall that we assigned the missions to each participant in a random order, making the order of missions differ from one participant to another. Fig. 10 shows the average time required to complete each mission by two groups: participants who carried out the mission as the first mission and participants who carried out the mission as the last mission. $First_i$ denotes the average time of those who started with mission i, and $Last_i$ indicates the average time of those who

performed mission i as their last mission. Interestingly, for each mission i, the average time of $Last_i$ is smaller than $First_i$, which means that participants tend to obtain a better understanding with the last mission. Notably, the difference in average time between $First_i$ and $Last_i$ is more noticeable for more complex missions (Wink$_{\{2,2\}}$ and SmartThings$_{\{2,2\}}$) that involved more conditions than the other missions. This shows participants quickly gained familiarity with Lumos.

VI. Related Work

IoT architectures for device interoperability: Many studies [2, 3, 4, 6, 8, 13, 18, 22, 45] present middleware-based approaches to improve interoperation among heterogeneous IoT devices. Some of the approaches [2, 3] leverage a smartphone as a mobile gateway and build middleware on top of the gateway. They offer REST APIs that write and load data from heterogeneous IoT devices. However, all approaches in this category necessarily require additional engineering effort from vendors. In contrast, Lumos focuses on supporting interoperation with no vendor support.

Program analysis: We build on the prior work that uses static analysis for network protocols [5, 9, 10, 11, 12, 15, 16, 17, 33, 41, 49, 63, 64]. These approaches analyze and detect network signatures, protocol state machines, etc. Dispatcher [9] reverse engineers a botnet's command-and-control (C&C) protocol. Replayer [49] and Rosetta [10] focus on re-playing application dialogues, while others [12, 15, 17, 41] aim to identify protocol fields within a message. In particular, like Lumos, Reformat [64], Extractocol [33], and ProDecoder [63] extract network message formats by inspecting application binaries. However, the main difference is that Lumos does not simply analyze network protocols but identifies the relationships between UI components and network behavior.

Programming by Demonstration (PBD): Programming by demonstration (PBD) is a useful technique to allow end users to automate their UI actions without requiring any programming [14, 34, 36, 37, 38, 39]. EPIDOSITE [37] focuses on mitigating interoperability problem of IoT devices by using the Android Accessibility API. The limitation of using the accessibility API is that an IoT app must be in the foreground whenever EPIDOSITE needs to detect a condition or issue a control action. In contrast, Lumos avoids this limitation as we focus on monitoring and emulating network level interactions between IoT apps and devices.

VII. Limitations

Legality and user privacy: As discussed throughout the paper, Lumos analyzes smart-home apps and control devices. Thus, if the license terms of a smart home app or device expressly prohibits any of the required levels of access by Lumos (e.g., if license terms prohibit device control by third-party software), Lumos cannot be used. Note that Lumos is designed to avoid extracting private information from apps. However, Lumos's control messages may contain private information when sending them for interoperation. We can extend Lumos to provide message inspection functionality for tech-savvy users to check their apps' messages for privacy.

Generality: Lumos has two limitations with regard to supporting various mobile apps. 1) Lumos-app cannot record actions for web-based apps due to the limitation of the Android accessibility mechanism. Incorporating HTML DOM accessing tools [7, 35] helps support such web-based apps. 2) Lumos cannot enable interoperation with non-Android devices as it analyzes Android apps only. This limitation can be remedied by a user who can manually inspect the network traffic of these devices.

User study: Note that our user study participants are university students, and this does not represent the general user demographic. However, the experimental results indicate a promising possibility of the adoption of Lumos by the general public. We believe that the richer and more interactive UI of Lumos will help increase usability, thus facilitating further adoption of Lumos.

Beyond message replay: Currently, Lumos replays previous messages (apart from updating them with the latest token). In our evaluation, we show that this technique alone enables the programming of many practical use cases. However, this simple message replay can be improved with fine-grained semantic information, such as the semantic meanings of individual fields, which would bring greater flexibility to devising diverse scenarios. (e.g., waiting for three mins after the door is locked before turning off the fan).

VIII. Conclusion

Despite the development of large IoT platforms and industry efforts for interoperability, we observe that the ecosystem is still largely fragmented. We posit that efforts that rely on vendor support or architectural changes face a fundamental challenge in deployment. To overcome the problem, Lumos takes a best-effort approach that leverages information embedded in IoT apps and combines it with semantic information from users. Our evaluation across 15 IoT devices from popular vendors demonstrates the feasibility of the approach. In particular, our user study with 24 participants demonstrates that Lumos-app offers a usable programming tool that enables the interoperation of IoT devices, offers interoperation features that are not available on commodity IoT platforms, and requires reasonable effort compared to three popular IoT platform-native apps.

IX. Acknowledgement

We thank our shepherd, Ralph Holz, for helpful comments. This work was supported by Institute of Information & communications Technology Planning & Evaluation (IITP) funded by the Korea government (No. 2020-0-00191, Research and Development of Modularized Library and Engine for Blockchain Emulation and Testing), (No. 2015-0-00164, Creation of PEP based on automatic protocol behavior analysis and Resource management for hyper connected for IoT Services); and the National Research Foundation of Korea (NRF) funded by the Korea government [NRF-2017H1A2A1042363].

References

[1] AllJoyn. an open source software framework that makes it easy for devices and apps to discover and communicate with each other. https://openconnectivity.org/developer/reference-implementation/alljoyn, 2019.

[2] G. Aloi, G. Caliciuri, G. Fortino, R. Gravina, P. Pace, W. Russo, and C. Savaglio. A mobile multi-technology gateway to enable IoT interoperability. In *IEEE First International Conference on Internet-of-Things Design and Implementation (IoTDI)*, pages 259–264. IEEE, 2016.

[3] G. Aloi, G. Caliciuri, G. Fortino, R. Gravina, P. Pace, W. Russo, and C. Savaglio. Enabling IoT interoperability through opportunistic smartphone-based mobile gateways. *Journal of Network and Computer Applications*, 81:74–84, 2017.

[4] A. Badii, J. Khan, M. Crouch, and S. Zickau. Hydra: Networked embedded system middleware for heterogeneous physical devices in a distributed architecture. In *Final External Developers Workshops Teaching Materials*, page 4, 2010.

[5] I. Bermudez, A. Tongaonkar, M. Iliofotou, M. Mellia, and M. M. Munafò. Towards automatic protocol field inference. *Computer Communications*, 84:40–51, 2016.

[6] M. Blackstock and R. Lea. IoT Interoperability: A Hub-based Approach. In *International Conference on the Internet of Things (IoT)*, pages 79–84. IEEE, 2014.

[7] M. Bolin, M. Webber, P. Rha, T. Wilson, and R. C. Miller. Automation and customization of rendered web pages. In *Proceedings of the 18th annual ACM symposium on User interface software and technology*, pages 163–172, 2005.

[8] A. Bröring, S. Schmid, C.-K. Schindhelm, A. Khelil, S. Kabisch, D. Kramer, D. Le Phuoc, J. Mitic, D. Anicic, and E. Teniente López. Enabling IoT Ecosystems through Platform Interoperability. *IEEE software*, 34(1):54–61, 2017.

[9] J. Caballero, P. Poosankam, C. Kreibich, and D. Song. Dispatcher: Enabling Active Botnet Infiltration usingAutomatic Protocol Reverse-Engineering. In *Proceedings of the 16th ACM conference on Computer and communications security*, pages 621–634. ACM, 2009.

[10] J. Caballero and D. Song. Rosetta: Extracting protocol semantics using binary analysis with applications to protocol replay and natrewriting. *CyLab*, page 32, 2007.

[11] J. Caballero and D. Song. Automatic Protocol Reverse-Engineering: Message Format Extraction and Field Semantics Inference. *Computer Networks*, 57(2):451–474, 2013.

[12] J. Caballero, H. Yin, Z. Liang, and D. Song. Polyglot: Automatic extraction of protocol message format using dynamic binary analysis. In *Proceedings of the 14th ACM conference on Computer and communications security*, pages 317–329, 2007.

[13] M. Caporuscio, P.-G. Raverdy, and V. Issarny. ubiSOAP: A Service-Oriented Middleware for Ubiquitous Networking. *IEEE Transactions on Services Computing*, 5(1):86–98, 2012.

[14] J.-H. Chen and D. S. Weld. Recovering from Errors during Programming by Demonstration. In *Proceedings of the 13th international conference on Intelligent user interfaces*, pages 159–168. ACM, 2008.

[15] P. M. Comparetti, G. Wondracek, C. Kruegel, and E. Kirda. Prospex: Protocol Specification Extraction. In *Security and Privacy, IEEE Symposium on*, pages 110–125. IEEE, 2009.

[16] W. Cui, V. Paxson, N. Weaver, and R. H. Katz. Protocol-Independent Adaptive Replay of Application Dialog. In *Proceedings of the Network and Distributed System Security Symposium (NDSS)*, 2006.

[17] W. Cui, M. Peinado, K. Chen, H. J. Wang, and L. Irun-Briz. Tupni: Automatic reverse engineering of input formats. In *Proceedings of the 15th ACM conference on Computer and communications security*, pages 391–402, 2008.

[18] P. Desai, A. Sheth, and P. Anantharam. Semantic Gateway as a Service Architecture for IoT Interoperability. In *IEEE International Conference on Mobile Services*, pages 313–319. IEEE, 2015.

[19] M. Díaz, C. Martín, and B. Rubio. State-of-the-art, challenges, and open issues in the integration of Internet of things and cloud computing. *Journal of Network and Computer applications*, 67:99–117, 2016.

[20] Z. Durumeric, Z. Ma, D. Springall, R. Barnes, N. Sullivan, E. Bursztein, M. Bailey, J. A. Halderman, and V. Paxson. The Security Impact of HTTPS Interception. In *Proceedings of the Network and Distributed System Security Symposium (NDSS)*, 2017.

[21] M. Elkhodr, S. Shahrestani, and H. Cheung. The Internet of Things: New Interoperability, Management and Security Challenges. *arXiv preprint arXiv:1604.04824*, 2016.

[22] K. Gama, L. Touseau, and D. Donsez. Combining heterogeneous service technologies for building an Internet of Things middleware. *Computer Communications*, 35(4):405–417, 2012.

[23] Google. Chromecast is a line of digital media players developed by Google. https://store.google.com/product/chromecast, 2019.

[24] IFTTT. The free way to get all your apps and devices talking to each other. https://ifttt.com/, 2019.

[25] Insteon. Creating a Scene with the Insteon Hub (Android). https://www.insteon.com/support-knowledgebase/2015/7/7/creating-a-scene-with-the-insteon-hub-android, 2019.

[26] M. Intelligence. Smart Homes Market - Growth, Trends, and Forecast (2019 - 2024). https://www.mordorintelligence.com/industry-reports/global-smart-homes-market-industry, 2018.

[27] IoT Agenda. A smart home app is an application used to remotely control and manage connected non-computing devices in the home, typically from a smartphone or tablet. https://internetofthingsagenda.techtarget.com/definition/smart-home-app-home-automation-app, 2015.

[28] IoT Agenda. Why interoperability holds the keys to the smart home. https://internetofthingsagenda.techtarget.com/blog/IoT-Agenda/Why-interoperability-holds-the-keys-to-the-smart-home, 2016.

[29] IoT analytics. Iot platforms company list 2017. https://iot-analytics.com/iot-platforms-company-list-2017-update/, 2017.

[30] IoT for all. Smart Home Interoperability: The Key Hurdles. https://www.iotforall.com/smart-home-interoperability-key-hurdles/, 2019.

[31] IoTivity. An open source software framework enabling seamless device-to-device connectivity to address the emerging needs of the Internet of Things. https://iotivity.org/, 2019.

[32] J. Kiljander, A. D'elia, F. Morandi, P. Hyttinen, J. Takalo-Mattila, A. Ylisaukko-Oja, J.-P. Soininen, and T. S. Cinotti. Semantic interoperability architecture for pervasive computing and internet of things. *IEEE access*, 2:856–873, 2014.

[33] J. Kim, H. Choi, H. Namkung, W. Choi, B. Choi, H. Hong, Y. Kim, J. Lee, and D. Han. Enabling Automatic Protocol Behavior Analysis for Android Applications. In *Proceedings of the 12th International on Conference on emerging Networking EXperiments and Technologies*, pages 281–295. ACM, 2016.

[34] T. A. Lau and D. S. Weld. Programming by Demonstration:An Inductive Learning Formulation. In *Proceedings of the 4th international conference on Intelligent user interfaces*, pages 145–152. ACM, 1998.

[35] G. Leshed, E. M. Haber, T. Matthews, and T. Lau. Coscripter: automating & sharing how-to knowledge in the enterprise. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, pages 1719–1728, 2008.

[36] T. J.-J. Li, A. Azaria, and B. A. Myers. SUGILITE: Creating Multimodal Smartphone Automation by Demonstration. In *Proceedings of the 2017 CHI Conference on Human Factors in Computing Systems*, pages 6038–6049. ACM, 2017.

[37] T. J.-J. Li, Y. Li, F. Chen, and B. A. Myers. Programming IoT Devices by Demonstration Using Mobile Apps. In *International Symposium on End User Development*, pages 3–17. Springer, 2017.

[38] T. J.-J. Li and O. Riva. Kite: Building Conversational Bots from Mobile Apps. In *Proceedings of the 16th Annual International Conference on Mobile Systems, Applications, and Services*, pages 96–109. ACM, 2018.

[39] H. Lieberman. *Your Wish is My Command: Programming By Example*. Morgan Kaufmann, 2001.

[40] Lifewire. What Is Google Brillo and Weave? https://www.lifewire.com/what-is-brillo-weave-1616282, 2019.

[41] Z. Lin, X. Jiang, D. Xu, and X. Zhang. Automatic protocol format reverse engineering through context-aware monitored execution. In *NDSS*, volume 8, pages 1–15, 2008.

[42] Microsoft. OpenT2T github. https://github.com/openT2T/translators, 2015.

[43] Microsoft. OpenT2T - An Open Source project to translate common IoT schemas to specific hardware devices. https://www.opentranslatorstothings.org/, 2019.

[44] J. Mineraud, O. Mazhelis, X. Su, and S. Tarkoma. A gap analysis of Internet-of-Things platforms. *Computer Communications*, 89:5–16, 2016.

[45] J. Mineraud and S. Tarkoma. Toward interoperability for the Internet of Things with meta-hubs. *arXiv preprint arXiv:1511.08063*, 2015.

[46] D. Miorandi, S. Sicari, F. De Pellegrini, and I. Chlamtac. Internet of things: Vision, applications and research challenges. *Ad hoc networks*, 10(7):1497–1516, 2012.

[47] D. Naylor, R. Li, C. Gkantsidis, T. Karagiannis, and P. Steenkiste. And Then There Were More: Secure Communication for More Than Two Parties. In *Proceedings of the 13th International Conference on emerging Networking EXperiments and Technologies*, pages 88–100. ACM, 2017.

[48] D. Naylor, K. Schomp, M. Varvello, I. Leontiadis, J. Blackburn, D. R. López, K. Papagiannaki, P. Rodriguez Rodriguez, and P. Steenkiste. Multi-context TLS (mcTLS): Enabling secure in-network functionality in TLS. In *ACM SIGCOMM Computer Communication Review*, volume 45, pages 199–212. ACM, 2015.

[49] J. Newsome, D. Brumley, J. Franklin, and D. Song. Replayer: Automatic protocol replay by binary analysis. In *Proceedings of the 13th ACM conference on Computer and communications security*, pages 311–321, 2006.

[50] M. Noura, M. Atiquzzaman, and M. Gaedke. Interoperability in internet of things: Taxonomies and open challenges. *Mobile Networks and Applications*, 24(3):796–809, 2019.

[51] Reddit. Homeautomation forum. https://www.reddit.com/r/HomeAutomation, 2019.

[52] ResearchAndMarkets.com. Smart Home Market by Product, Software & Services, and Region - Global Forecast to 2024. https://www.researchandmarkets.com/research/r4xlbj/smart-home-market, 2019.

[53] Sam Solutions. A Mobile App for a Smart Home Solution: Takeaways and Guidelines. https://www.sam-solutions.com/blog/why-is-a-mobile-app-important-for-an-efficient-smart-home-solution/, 2019.

[54] Samsung. SmartThings devices-integrations forum. https://community.smartthings.com/c/devices-integrations, 2019.

[55] SmartThings. SmartThings hub stores the latest value locally. https://community.smartthings.com/t/does-device-latestvalue-query-the-device-or-just-the-hub-battery-usage-concerns/61228, 2016.

[56] SmartThings. Developer Documentation. = https://smartthings.developer.samsung.com/docs/index.html, 2019.

[57] SmartThings. Publish your devices to Smart-

Things. https://smartthings.developer.samsung.com/distribution, 2019.

[58] SmartThings. Support Products. https://www.smartthings.com/products, 2019.

[59] SmartThings. Web Service. https://docs.smartthings.com/en/latest/smartapp-web-services-developers-guide/overview.html, 2019.

[60] Z. Song, A. A. Cárdenas, and R. Masuoka. Semantic middleware for the internet of things. In *Internet of Things (IoT)*, pages 1–8. IEEE, 2010.

[61] TechTarget. The IoT journey for manufacturers: concept, production and beyond. https://internetofthingsagenda.techtarget.com/blog/IoT-Agenda/The-IoT-journey-for-manufacturers-Concept-production-and-beyond, 2017.

[62] TLSeminar. TLS Interception and SSL Inspection. https://tlseminar.github.io/tls-interception/, 2017.

[63] Y. Wang, X. Yun, M. Z. Shafiq, L. Wang, A. X. Liu, Z. Zhang, D. Yao, Y. Zhang, and L. Guo. A Semantics Aware Approach toAutomated Reverse Engineering Unknown Protocols. In *IEEE International Conference on Network Protocols (ICNP)*, pages 1–10. IEEE, 2012.

[64] Z. Wang, X. Jiang, W. Cui, X. Wang, and M. Grace. ReFormat: Automatic Reverse Engineering ofEncrypted Messages. In *European Symposium on Research in Computer Security*, pages 200–215. Springer, 2009.

[65] A. Whitmore, A. Agarwal, and L. Da Xu. The Internet of Things - A survey of topics and trends. *Information Systems Frontiers*, 17(2):261–274, 2015.

[66] Wink Labs. Wink Door/Window sensor manual. http://manuals-backend.z-wave.info/make.php?lang=en&sku=Wink%20D/W%20Sensor&cert=ZC10-17075685, 2019.

[67] Wink Labs. Wink support products. https://www.wink.com/help/products, 2019.

[68] C. Yang, B. Yuan, Y. Tian, Z. Feng, and W. Mao. A Smart Home Architecture Based on Resource Name Service. In *2014 IEEE 17th International Conference on Computational Science and Engineering*, pages 1915–1920. IEEE, 2014.

[69] T. Zachariah, N. Klugman, B. Campbell, J. Adkins, N. Jackson, and P. Dutta. The Internet of Things Has a Gateway Problem. In *Proceedings of the 16th international workshop on mobile computing systems and applications*, pages 27–32. ACM, 2015.