Received November 11, 2021, accepted December 12, 2021, date of publication January 19, 2022, date of current version February 11, 2022.

Digital Object Identifier 10.1109/ACCESS.2022.3144913

BULWARK: A Framework to Store IoT Data in User Accounts

JEREMY LYNN REED[1] AND ALI ŞAMAN TOSUN[2]
[1]Department of Computer Science, The University of Texas at San Antonio, San Antonio, TX 78249, USA
[2]Department of Mathematics and Computer Science, The University of North Carolina at Pembroke, Pembroke, NC 28372, USA

Corresponding author: Jeremy Lynn Reed (jeremy.reed@my.utsa.edu)

ABSTRACT The explosive growth of the Internet of Things (IoT) devices raises serious concerns for a user's privacy and security because the existing software framework on these devices often support various default features and generate large data sets. Moreover, many IoT devices incorporate a manufacturer-owned cloud-based back-end support to process and store the generated data while simultaneously sharing with third parties. Clearly, in such an industry-driven environment with the desire to use the IoT data as a revenue stream, it is a challenge for users to control IoT data. Device manufacturers utilize an opaque software design where user data is generated and stored with little transparency. Manufacturers use EULAs as a legal construct to protect a manufacturer's legal standing and to explain a device's behavior, however this explanation is vague and lacks the necessary details for a user to determine a device's acceptable use and it has become increasingly difficult for users to secure and maintain their data. Fortunately, as the privacy minded user base of IoT devices grows, the manufacturers will be forced to implement a new framework that can enable users to have more control on the creation of their IoT data, and to store/disseminate such data in a secure and private manner. In this paper, we address this lack of transparency from manufacturers and address the issues of privacy and security by proposing a new framework called Bulwark, for manufacturer use on IoT devices and mobile applications. Proposed framework enables the user to generate and manage a set of data controlling rules, and store the result in their personal cloud account, while providing a dashboard data reporting tool enabling data transparency and supporting good user choices. The user's ability to access, disseminate and secure IoT generated data, is now available within our proposed framework. Using reverse engineering, simulation and implementation of open source solutions, we demonstrate support for a set of common devices. Each device executed the framework, while communicating with a mobile application and cloud services. Rules were generated for each message and telemetry was returned to the mobile application for dashboard rendering. We stored generated data in the cloud using our own account, while maintaining the free tier for each of the cloud services. Network usage increased between 4% and 9% while storage size grew between 0% and 2% larger, as compared to using the device without the framework. Our framework demonstrates support for a multitude of devices, by either open source or support for similar feature sets. This framework is easy to integrate and we anticipate wide spread adoption.

INDEX TERMS IoT security, IoT privacy, cloud computing.

I. INTRODUCTION

Data privacy has received a lot of attention recently [1] driven by the explosion of new areas of data generation and aggregation. With the advent of social networks and search engines, users are able to interact with the Internet in new and novel ways. Social networks give access to an unprecedented number of people while search engines translate queries into aggregated data. Both genres of software open a flood gate to user data. For the first time we can observe social interaction on a massive scale and data mine this behavior. IoT devices were designed as a natural extension of this evolution. Instead of limiting a user's interaction to a web browser, IoT devices were created to directly interface social media. Mobiles applications enabled access to photos, video and GPS features. Customized devices such as Siri and Alexa directly couple home automation and home security to their respective manufacturers. This expansion of function is not limited to

The associate editor coordinating the review of this manuscript and approving it for publication was Nadeem Iqbal.

compute devices as the IoT model expanded into every facet of our lives. Cloud based architectures for IoT devices while commonplace, raise numerous privacy concerns. Since IoT sensor data often embeds private user information, sending this data to the cloud service implies that the cloud analytic service has the ability to extract and process this information. In many cases, the embedded information is not related to the original purpose for which the data was collected, and can result in data disclosure with unintended consequences. As an example consider the recent privacy issue introduced when Strava, a popular fitness tracking app used by runners, published anonymous heat maps of popular running routes. Even though personal data was scrubbed from the application and the generated maps, the location data contained in the running routes inadvertently revealed the locations of secret military bases [2]. In short, it was easy to extrapolate who was running in the desert by location, frequency and route. Maintaining user privacy in the face of cloud-based IoT services is a challenging problem. One approach is to use IoT devices ''locally'' without the benefit of a back-end cloud service. This implies a user lacks control of their IoT devices while not connected to a home network, limiting the functionality of a device. IoT devices represent a concern for older users, as the difference between devices which have occupied their homes for many years and their corresponding ''smart'' iteration is not obvious. For example, smart refrigerators track device usage, generate shopping lists and translate each list to a supermarket order, however the functionality needed to generate an accurate list and communicate with a grocer's API may not be obvious to the average consumer. Cameras are needed to observe the contents and remote computing is used to discern food type and quantity. An account is used to correlate this data to a grocer's account and historical food consumption is stored with both the refrigerator account vendor, as well as the grocer account. It is this disconnect in device function that presents a challenge to the security and privacy of user data. It is projected that more than 100 billion devices will be Internet-connected by 2025 [3], and with a current value of $267 billion per year [4], an investment of security spending at about 1% [5] is under funding the problem. Given the lack of security engagement, we chose to tackle this problem as an opportunity to build new research around these devices. Our work focuses on storing IoT data in cloud services user accounts such that a user gains control over the creation, dissemination and storage of their own data. Figure 1 (left) represents a common network for a manufacturer's solution. The device is executing a closed source solution which brokers data between the mobile application and the cloud. The user is required to register an account with the device manufacturer, which serves to identify the user, while the underlying cloud services account is not disclosed by the device manufacturer and serves as a single account storage repository. Third-party services are implemented at the device manufacturer's discretion, often not disclosed in a clear manner to the consumer. Figure 1 (right) represents our vision of a secure and transparent IoT network solution. By controlling data routing, we control when data is generated, how it propagates through the network and where it is stored in the cloud. Our framework is called BULWARK as it serves to build a wall between user data and non user software. Contributions of the paper are as follows.

- We propose a framework which stores IoT data in a user's chosen cloud account and provide the user full control of the data.
- The proposed framework has many desirable features including easy to use interface, easily scales as device support grows, low cost, data transparency, secure data storage and enforceable rules.
- The proposed framework is extensible to a wide range of devices, reducing complexity of development.
- Cost of the framework falls under the free tier of all major cloud service providers.
- Using the framework benefits device manufacturers by eliminating the need to support cloud function for IoT devices.

The rest of the paper is organized as follows: In section II, we describe related work. Section III presents motivation and section IV explains desirable features of such a framework. Proposed framework is explained in section V and experimental results are described in section VI. We discuss various issues in section VII, framework extensions in section VIII and conclude with section IX.

II. RELATED WORK

In this section, we describe related work on various aspects of IoT systems including networking, security, data collection, processing, device identification and practical implementation issues.

Various aspects of IoT Networking have been investigated in the literature. Unlike traditional networked applications or devices like a web browser or a PC, IoT applications and devices serve narrowly defined purposes and do not require access to all services in the network. Therefore, IoT device communications should be default-off and desired network communications must be explicitly enabled [6]. Resilient overlays for reliable delivery of sensor data from IoT devices to distributed cloud service instances in the face of localized failures for event detection, community infrastructure management, and emergency response is proposed [7]. One-to-many data transmission in smart devices at close range is investigated in [8]. Prior methods require the use of an extra application service where the operating system differs between smart devices. In contrast, the proposed method makes use of the smart device's built-in speaker and microphone to confirm the transmission signal. Data is then transmitted via Wi-Fi or long term evolution. A scalable and adaptive model which efficiently and quickly enforces control schemes for IoT via a novel command messaging service is proposed [9]. It is achieved by utilizing the n-tier scalability of the cloud to generate vast networks of virtual machines. Smart home IoT traffic is characterized in terms of its volume, temporal patterns, and external endpoints along with focusing on certain security and privacy concerns [10].

IoT security has received a lot of attention recently. LogSafe is a scalable, fault-tolerant logger that leverages the use of Intel Software Guard Extensions (SGX) to store logs from IoT devices efficiently and securely [11]. A utility-preserving privacy technique, which intelligently obfuscates smart energy meter data to prevent leaking a home's private occupancy information, while retaining the ability to perform useful energy disaggregation analytics is proposed [12]. An open-source toolkit for construction and deployment of an authorization service infrastructure for IoTs is proposed [13]. The infrastructure uses distributed local authorization entities, which provide authorization services that can address heterogeneous security requirements and resource constraints in the IoT. Challenges of the IoT data communication framework with authenticity and propose two approaches called Dynamic Tree Chaining and Geometric Star Chaining that provide authenticity, integrity, sampling uniformity, system efficiency, and application flexibility to IoT data communication are addressed [14]. The problem of efficiently and effectively securing IoT networks by carefully allocating security tools is the focus of [15]. They model the problem using game theory, and provide a Pareto-optimal solution, in which the cost of the security infrastructure, its energy consumption, and the probability of a successful attack, are minimized.

Data collection and processing is another core issue of IoT systems and attracted a lot of attention. Tethys [16] crowdsources the data collection process to residents' smartphones acting as gateways. These gateways are untrusted and unreliable, so Tethys implements end-to-end reliability and security between the sensing device and a cloud backend. An industrial IoT architectural framework that allows data offloading between the cloud and the edge is proposed [17]. This framework is used for telemetry of a set of heterogeneous sensors attached to a scale replica of an industrial assembly plant. The use of building Wi-Fi data for quantitative evaluations of both planned and unplanned evacuation events is explored [18]. Query processing for IoTs is investigated and analytic conditions for the optimal coupling between the device energy consumption and the incurred cloud infrastructure billing is derived [19]. In a related paper, an analysis of standby energy problem is provided, and four primary issues that contribute to the standby energy problem based on a study of commercially available smart devices are identified [20].

Device association and identification have also been investigated extensively. New approaches take advantage of ubiquitous light sources around to perform continuous device grouping based on the similarity of light signals [21]. To control the spatial granularity of user's proximity, it provides a configuration framework to manage the lighting infrastructure through customized visible light communication. IoT-ID, a device-specific identifier, that captures the device characteristics and can be used towards device identification [22]. IoT-ID is based on physically unclonable functions (PUFs), that exploit variations in the manufacturing process to derive a unique fingerprint for integrated circuits. A calibration-free passive sensing approach that utilizes human-device motion to determine the user, body location of each device is proposed [23].

Recently, approaches based on machine learning are applied to IoT systems. The impact of timing errors on a multimodal fusion classifier for human activity recognition is quantified [24]. For compact CNN models, it is challenging to sustain high inference performance due to limited and varying inter-device bandwidth. A streaming inference framework to simultaneously improve throughput and accuracy by communication compression is proposed [25].

Edge computing support for IoT, fault tolerance and testbed design are practical aspects of the work done on IoT systems. Data replication strategies and a real-time and fault-tolerant edge computing architecture for IoT applications are investigated [26]. ECCO [27] is an orchestration framework that enables edge-cloud collaborative computing for road context assessment to detect and react efficiently to road hazards. IoTREPAIR [28] is a fault-handling system for IoT that integrates with fault identification modules to track faulty devices, provides a library of fault-handling functions for effectively handling different fault types and provides a fault handler on top of the library for autonomous IoT fault handling, with deployed devices, user preferences, and developer configuration as input. LinkLab [29] is a scalable IoT testbed for heterogeneous devices that not only supports running experiments but also supports remote development via a web-based IDE and remote compiling.

Other relevant work includes design and development issues including architecture, modular approaches and design patterns. Horizontal integration, where sensors and actuators from different applications can interconnect with any computational IoT application is investigated and a layered protocol architecture for scalable innovation and identification of network economic synergies in IoTs is proposed [30]. IoT applications are generally black-box, end-to-end application-specific implementations, and cannot keep up with timely resolution of all this live, continually updated, heterogeneous data. A modular approach to context-aware applications, breaking down monolithic applications into an equivalent set of functional units, or context engines is proposed [31]. By exploiting the characteristics of context-aware applications, context engines can reduce compute redundancy and computational complexity. A wide range of design disciplines involved in creating IoT systems, that act as a seamless interface for collaborating heterogeneous things, and suitable to be implemented on resource-constrained devices are identified [32]. The IoT patterns covered vary in their granularity and level of abstraction. They are inter-related, well-structured design artifacts, providing efficient and reliable solutions to recurring problems discovered by IoT system architects.

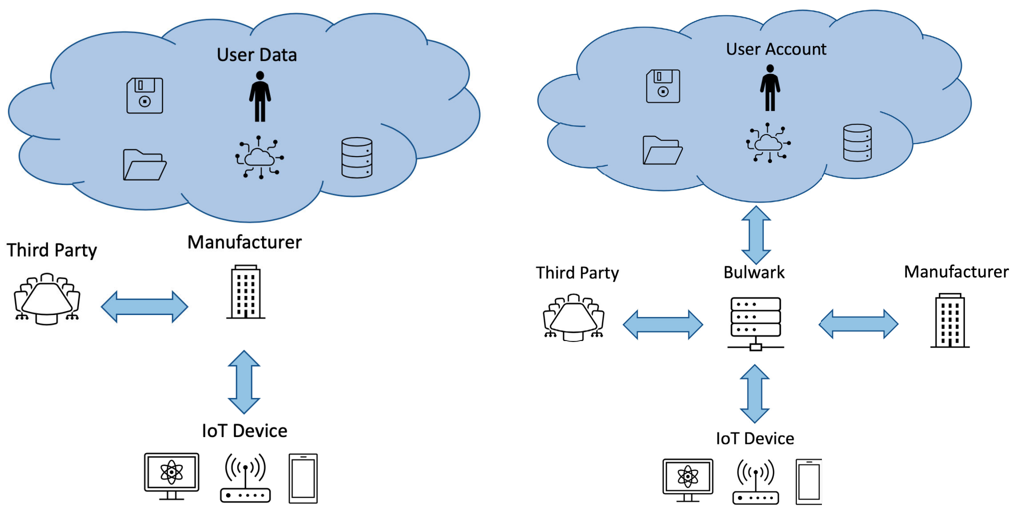

FIGURE 1. Comparison of Default and Bulwark Environment.

III. MOTIVATION IN PROVIDING A SECURE AND PRIVATE FRAMEWORK

An examination of IoT manufacturing practices has yielded a common thread in device behavior. In almost every device examined, back end storage and processing via cloud services was used to enable remote functionality. When considering a small set of devices allowing users to opt out of remote features, a majority of these devices continued to communicate with a cloud provider. It is a concerning trend when incorporating Internet connectivity into everyday products requires remote storage and the dissemination of user data. As an example, consider the many consumers who have purchased IoT enabled refrigerators and are enjoying the remote capabilities of the device, while lacking a fundamental understanding of the included cloud services. Realizing the excitement of issuing a command to an appliance and have it respond accordingly, a user's satisfaction might be tempered when it becomes clear that their refrigerator contains an always on always listening remotely connected microphone. In the course of researching the subject, we observed a fundamental disconnect between a user's expectation of device behavior and actual observed behavior. We believe this disconnect is driven by years of interaction with non-networked devices and developing an implicit expectation of device behavior and carrying over this assumption to the IoT versions.

Another example of this disconnect is the introduction of computing within automobiles. For many, the idea of combining an automobile with a concierge service and computing resources represents a paradigm shift in vehicle interaction. We forego paper maps and instead speak directly with remotely connected humans, consult in car and Internet based navigation services, and utilize an Internet connection for a wide range of features. Vehicles guide us to our destination, provide luxury services such as setting restaurant reservations, and connect us with emergency medical services as needed. Introduce a mobile application and the autonomous features expand further. Locating a vehicle is trivial using nothing more than a phone and mapping software. Property security is increased as mobile apps report alarm activation, vehicle movement and real time location data. Considering these features, how many owners have considered the scope of intrusion from multiple parties or the generated data needed to support these services. To frame the question in clearer terms, are we comfortable with automobile manufacturers and third party vendors accessing real-time audio, video and data. Communication of route and speed is a necessary component for many of the outlined features, yet most people feel that operating an automobile is an relatively anonymous process. The result of an always on microphone and camera, coupled with real time data generation, is real time tracking.

Manufacturers acknowledge sharing aggregated data with third parties and law enforcement, but stop short of clarifying the data type and usage [33]. An industry has emerged which caters to use formal aggregation of device data for the purposes of criminal investigation [34]. One often asked question is "Can generated data serve as a foundation for prosecuting a crime?" Not only is this data used in criminal prosecution, there are companies devoted to facilitating this behavior [35]. These are the types of questions which raise a red flag concerning the proliferation of IoT devices. We also consider the issue that the current application of these features is based on legal precedent and social norms, both of which change over time. Based on this broad examination we propose a set of boundaries, representing a clear delineation between a user's privacy and security, and the manufacturer's desire to include a user's data as a revenue stream. The easiest method of enforcing these boundaries is enabling a user to maintain control of their data through the use of cloud account ownership and a set of clearly defined rules. We propose this change as a replacement for the current IoT device development process.

IV. DESIRABLE FEATURES OF AN IDEAL FRAMEWORK

An IoT framework supports communication between one or more IoT devices, a mobile application and cloud services. An ideal framework focuses on inserting boundaries into these features, such that a user's security and privacy are respected. This is in contrast to a policy based model where legal terms are defined within EULAS. EULAS represent among other policies, a legal stance on a user's data. Users can accept or decline a EULA, but not modify the terms. With this all or nothing approach, it's clear to see how the policy model favors device manufacturers. An ideal framework should assert a set of limits which enforce a balance between business needs and user rights. Desirable properties of an ideal framework include the following.

- **Ease of Use.** Cloud services and associated credentials are an additional layer of complexity to IoT ownership. An ideal framework streamlines the process for creating or using existing cloud credentials.
- **Scalability.** Pairing multiple devices to a single environment is possible. When pairing, the framework will recognize and reuse common parameters and reduce the pairing steps. Instead of an application and cloud service for each device, one environment serves a number of devices.
- **Cost.** The cost of storing one's data in the cloud should not be prohibitively expensive. Most cloud service providers offer a free tier. When possible, the cost of an ideal framework would fall under the free tier.
- **Data Transparency.** A user should be able to view and disseminate IoT generated data. An ideal framework provides the ability to observe the generation, transmission and storage of data in real time.
- **Secure Data Storage.** A core tenant of an ideal framework is the ability to remove a user's data from a manufacturer's access. Generating data within the framework and storing the result in a user's cloud account, ensures a secure end to end process of data generation, observation and storage.
- **Enforceable Rules.** Enforce the requisite boundaries using a rules engine. Rules allow a user to accept or reject device features, control data flow and lock down the creation and sharing of data.

V. PROPOSED FRAMEWORK

Using the tenants outlined in an ideal framework, we designed a framework to control data and empower the user with a set of choices. Our framework executes on the device, in the cloud and on the user's mobile device. The framework controls data flow with a permission, backed by corresponding rules. The mobile application renders the user's choices in a set of dashboards, clarifying the current environment and giving the user the ability to change their choices.

The framework design accounts for both user and manufacturer needs by separating execution into three regions. A modular approach is used through object oriented programming and design. Implementing a specific IoT device is as easy as defining a set of interfaces. The framework consumes these definitions and generates a set of dashboards and rules for use within the mobile application. The Java programming language was our choice for implementing our framework as it's object oriented approach and support for Remote Method Invocation (RMI) was a natural fit for an abstract solution. RMI also provides object versioning and a remote object repository, which solves many of the problems associated with intermittent network communication.

The framework executes in three environments. On the **IoT device** the framework executes an API to listen for commands and generates responses when appropriate. The commands are validated and compared against the current set of rules. If a command passes the verification process, it is translated into a native device command and executed. The **mobile application** is tasked with the initial configuration of the device and the cloud service. Once configuration is complete, the application will use three user interface panels to interface with the device and the cloud service. Panel one provides a user interface to control the device while panel two reports telemetry data from the device and the cloud. A third panel contains a set of rules generated by the framework and the appropriate user interface to set their respective values. The **cloud service** is a preconfigured environment, executing within the context of the user's cloud credentials. Data and telemetry is stored within the cloud and commands are brokered between the mobile application and the device enabling remote interaction.

A. MODULAR APPROACH TO IoT DEVICE SUPPORT

The framework is built on Java RMI technology, such that message passing between the device, the controlling application and the cloud backing service is implemented as remote objects. The framework defines a message schema and enforces a strict message shape. Beyond this schema, manufacturers are free to define meaningful messages based on the available framework types and the implemented IoT device.

In the Bulwark framework a **Message** represents the basic building block of a manufacturer's implementation. A message is an abstract Java RMI object, implementing *Serializable* and $Message < T >$. The schema enforces a strict message type and a dynamic data payload. The associated message type has constraints associated with it, which are used to enforce the user's permission set. A message is currently defined as follows as described in JSON format.

For example, consider a GPS enabled IoT device such as a drone. We could define the command GetLatitude, which returns the current devices' latitude value, with the response: "Double". The response implementation is API: getGPSLat, which returns the current device latitude. The portion of the framework executing on the user's application will query the permission set, for permission to call the GPS API. If a command is prohibited by the current permission set, the message is not sent. PKI is used to sign messages, such that messages injected on the network are ignored by the IoT device.

```
{"
    message": {"
        command": {"
            query": "String",
            "control": "String"
        },
        "response": {"
            state": {"
                type": ["Long", "Integer"
                , "Double", "String"]
            },
            "success": "boolean"
        }
    }
}
```

PKI data is generated on first pairing of the controlling application and the IoT device and stored in the cloud, which is controlled by the user.

Object Wrapper: Messages are wrapped within RMI objects and signed by the generating application. Commands are generated by the user via the controlling application. Responses are generated by the IoT device, and both types of messages can be created and received within the cloud framework. Messages are validated, wrapped, signed and classified as "Queue-able", or "real-time". Valid RMI objects are prioritized by class and delivered based on this priority. Expiry data is set within the RMI object, with default values for "real-time" as "deliver or expire" and "queue-able" as "cache and deliver." Cached commands are passed based on the order of generation and the responding application acknowledges the message by including the GUID. Messages which do not generate a response will receive an acknowledgement containing the message GUID.

1) CLOUD SERVICES

a: CONCRETE IMPLEMENTATION AND INHERITANCE

At the lowest level, each message contains data which is unique to the specific IoT device. For example, the getGPSLat implementation contains a formatted API call to the GPS hardware and wraps the device response in a response message. The message is formatted to only include the latitude and is then inserted into an RMI object, signed, stamped and executed. Given the previous steps, consider two disparate IoT devices. Device A and B contain differing hardware. The implementations differ only at the hardware definition level. Inheritance allows both messages to contain a single implementation, up to the hardware definition. This approach overcomes the challenge of reducing message size, publishing of prepared message templates, and for multiple implementations to share message components.

B. FRAMEWORK COMPONENTS

The framework is divided into three areas: The mobile application, Cloud services and the IoT device. The mobile application controls the IoT device and displays telemetry. The IoT device executes an API listener for message passing, verification and rules enforcement and cloud services is used to facilitate remote device interaction and storage.

1) IoT DEVICE

The device executes its own firmware and an instance of the framework. Within the framework we execute cloud telemetry, an instance of the authentication and rules engine and in some cases a cloud OS, as shown in Fig. 2. The device firmware is provided by the manufacturer and is used to control the device. Commands are passed to the device via the mobile application and the cloud. The command is parsed, authenticated and matched against a set of rules. if the command doesn't match a rule or a deny rule matches, the command fails with a telemetry entry. Entries are pushed to the cloud and utilized for postmortem debugging of anomalous behavior. Finally, framework commands are translated to firmware commands via the message implementation and the device executes the request. The telemetry engine represents a common cloud interface to measure, aggregate and store data. Measurement and aggregation is both controlled and reported by the rules engine. The results are sent to the both cloud and mobile application as telemetry data, where the mobile application renders the data to the user and cloud storage acts as a long term repository. A Cloud OS represents a unique configuration, where the user's preferred cloud supports the firmware implementation of the IoT device. In this case, a portion of the framework authentication and rules engine will execute within the cloud OS, authenticating commands and generating telemetry data. Communication between the device and the two counterpoints will continue to pass through the framework, but in this case the framework requests are processed in the cloud.

2) MOBILE APPLICATION

The mobile application contains the Command and Control and Telemetry and Rules panels, as shown in Fig. 3. Command and Control exposes user interface objects for device control and in the case of a smart switch, the panel contains a toggle switch and user interface power button. The switch toggles power and the user interface button lights up on power activation. Telemetry contains a set of reporting graphs, sharing real time and historical data. Historical data is aggregated from the cloud and real time usage is reported from the device. The rules panel contains rules for a given IoT device and the appropriate controls to set the rule. In the case of a smart switch the panel can contain a rule to allow the reporting of switch state to the cloud and to allow remote control, via any network. The first rule is a Boolean value and the second is a user interface dialog box representing the user's choice. An example value could be

```
localNetwork:Yes,
RemoteAny:No,
10.0.0.0/8:Yes
```

In the example rule, rule one allows remote control from any address on the local network. The second rule denies all

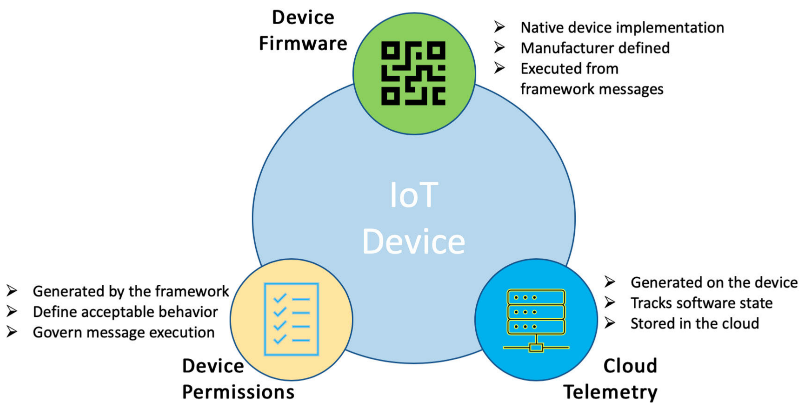

FIGURE 2. **IoT Device.**

remote control outside of the device's network and rule three allows remote control from all class A network addresses. Rules parsing stops at the first deny match, and otherwise parses to completion. In the example rule, a local network communication would parse as ''Yes'', as rule one allows the communication and rule two does not prohibit it. If the communicating network is remote and within a class A, the result would be disallow, as rule two prohibits all non local networks, regardless of rule three's value.

3) CLOUD SERVICES

As shown in Fig. 4, cloud services represent the remote compute and storage engine and is deployed as a single instance per cloud account. To clarify this work and highlight our challenges we will introduce an example implementation based on Terraform [36] and Amazon Web Services. We keep the solution abstract, allowing crossover function in other cloud providers, but use an Amazon Web Services example to avoid a vague discussion. The framework contains the initial web APIs for all three cloud services. After generating the shared secret and aggregating a user's cloud credentials, the mobile application references the appropriate cloud API and initiates authentication. In the case of Amazon Web Services, the user's credentials are used to determine the existence of an instance of an executing cloud framework. If none exists, Terraform is used to configure cloud objects. Initial deployment creates an instance of an Edge Lambda [37], a back end Lambda [38], an Elasticache instance [39], a database and an S3 bucket [40]. Each of these objects contain an internal network route enabling low latency inter-communication. A nodeJS script is deployed to each of the Lambdas to process requests.

a: EDGE LAMBDA

Executes at the edge of the cloud service and serves as the single point of contact for the framework solution. The lambda receives a request and unwraps the contents. The authentication token is verified as belonging to the user, the shared secret is verified as belonging to a single iOS device and the request is sanitized and validated for correctness. If the request does not require a response, or the response is served through caching, the Edge Lambda will respond via a callback and the work is complete. If the response requires further computation, the sanitized request is forwarded to the back end Lambda.

b: BACK END LAMBDA

Connects to an instance of Elasticache, a database and an S3 bucket. The purpose of this Lambda is to aggregate dependent data from either Elasticache, the database or an S3 bucket. The response is computed within the Lambda, cached within the Elasticache instance and simultaneously sent back to the mobile application and stored within the database and the S3 bucket.

c: ELASTICACHE INSTANCE

Stores the last nth% of cached responses, serving as an in memory cache. Responses lacking the ''no cache'' property and found within Elasticache are immediately returned to the back end Lambda, reducing response time to 10s of milliseconds.

d: S3 BUCKET

Used to store and return flat files, such as images and text. If a request requires a file, the file is pulled from the bucket, cached in Elasticache and returned to the back end Lambda.

e: DATABASE

Stores row entry items. We currently utilize the database for device state and logging. Each of the rows can be cached within the Elasticache instance, so long as the data is not notated with ''no-cache''. A common example would be storing the current state of a smart switch and caching the result. Subsequent state queries are returned from Elasticache, giving sub 10s of millisecond response times.

C. COMMUNICATION BETWEEN FRAMEWORK COMPONENTS

Reliable cross environment communication is necessary for a functioning framework and represents a core challenge.

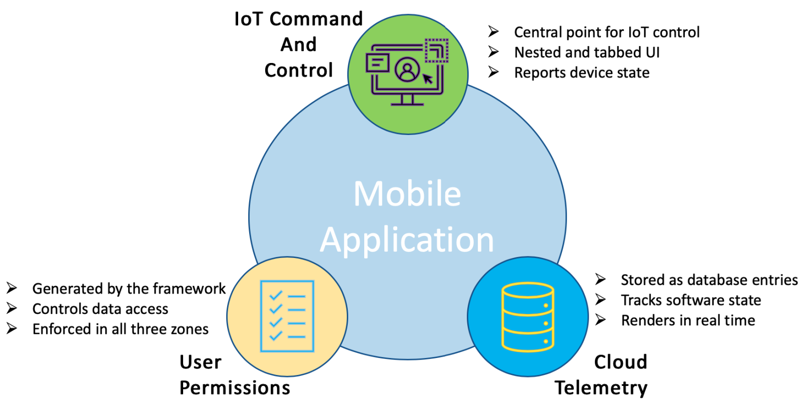

FIGURE 3. Mobile Application.

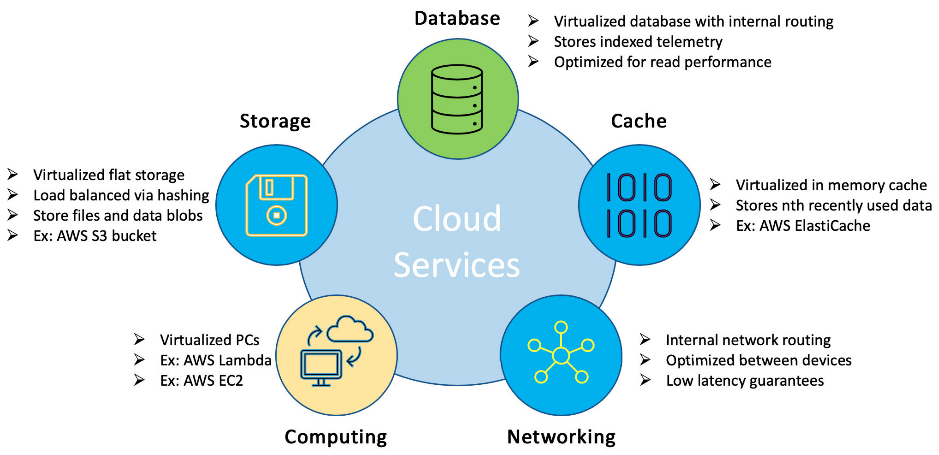

FIGURE 4. Cloud Services.

We introduced the use of local zone caching and a resilient fall back process as a means to combat intermittent network loss. Our broader design approach uses the mobile application as both a portal for human interaction and the initiating application for device setup. The device executes a portion of the framework for message authentication and rules enforcement, ensuring a user's permissions are respected. The cloud service is a remote layer between the application and the device, serving as a facilitator in interaction and storage. In the following subsections, we dive deeper into this design philosophy and highlight our solutions to common wireless IoT challenges.

The underlying technology for all network messaging is Java Remote Method Invocation (RMI). We chose RMI as it brings three key technical solutions: The remote registry, object versioning and the ability to supersede an implementation. The remote registry provides an implemented caching repository for messages. Message versioning is built into RMI object versioning, allowing us to effortlessly supersede older implementations with newer solutions. In short, messages are implementations of RMI objects. The object wraps a message type, which contains an implementation for message execution. Object fields store applicable rules and extraneous data needed to process a message. When a development team chooses to implement an IoT device the process is streamlined to defining and implementing RMI objects.

1) MOBILE APPLICATION AND CLOUD SERVICES

Using one our supported devices as an example, we walk through the communication process of a smart switch. A smart switch is a network enabled home light switch, used in home automation to remotely control lighting. In our framework, initial setup executes between the mobile application and the device. The user's cloud credentials are queried and the Terraform file is executed. Upon completion the shared secret and serial number are shared with the cloud and the device receives the cloud API from the mobile application. The mobile application is now paired to the device and the device is paired to the cloud, allowing the application to enter normal execution. Communication between the mobile application and cloud services is represented in the following steps:

1) Upon first run of the mobile application, a shared secret is generated and the user's cloud credentials are requested.
2) The mobile application contacts the provided cloud service and authenticates the user and the framework account. The framework cloud account contains a Terraform application file, which is used to deploy the framework cloud infrastructure as an instance within the user's account. The shared secret is stored in the cloud, which completes cloud deployment. In the case of an existing cloud infrastructure, the shared secret is stored for the new iOT device.
3) The mobile application will periodically request telemetry data from the cloud as initiated by the user or a programmatic process, as part of the telemetry dashboards.

2) MOBILE APPLICATION AND IoT DEVICE

Continuing our example of a smart switch, we focus on communication between the mobile application and the IoT device. The mobile application executes a multi-cast network request for the smart switch. Lacking a response, the mobile application blocks with the error that no device was found and repeats interval request. When a response is received, an IP address, sub-net mask and serial number are shared between device an application. The mobile application generates a shared secret using entropy, and the serial number. Cloud services setup executes and the resulting cloud API is shared with the device. Upon successful execution, the mobile application is now paired to the device and the device is paired to the cloud, allowing the application to enter normal execution.

1) The generated shared secret and cloud credentials are shared with the IoT device and the device returns the current set of feature rules. Feature rules represent a set of generated rules, created during device implementation. For example, a feature which queries a GPS radio will contain a set of corresponding rules providing user controlled limits of the feature. These rules are stored in the mobile application and a user interface panel is generated.
2) Once a user finalizes the device rules, the results are pushed to the device and stored as a set of hardware permissions. Deployment is complete and the device enters normal execution.

3) iOT DEVICE AND CLOUD SERVICES

a: EXAMPLE: SMART SWITCH

Upon completion of the mobile application and cloud services setup, the device contains the generated shared secret, the cloud API, cloud credentials and the user rules. Remote commands are received from the cloud service and acted upon appropriately. If an active connection exists between the mobile application and the device, the responses for the remote commands are mirrored to the application. Responses are generated and sent to the cloud for processing and storage. The device moves into a paired state, executing the following steps.

1) The device is dormant until such time as it receives the generated shared secret and cloud credentials.
2) Device rules are exchanged with the mobile application and the device pushes the rule results to the cloud for processing and storage.
3) Commands are received, validated and acted upon based on the current rule set. If a cloud response exists for a given rule, the response is packaged with the shared secret, credentials and timestamp, and sent to the cloud service for storage.
4) The IoT OS generates a response which is sent to both the mobile application and the cloud service. This data is used to track device interaction, health and provide an audit trail for postmortem analysis of device behavior.

D. RULE IMPLEMENTATION

A Bulwark rule is an object representing the result of a user permission. The rule is evaluated by the rules engine and a Boolean is returned. This result is used to enforce a user's choice. When a message is implemented, the manufacturer is in effect registering an RMI object and enlisting it into a set of permissions. For example, the *getGPSLat* message requires permission to access the GPS device. When the manufacturer implements this call, the framework automatically generates a set of user interface permissions, which are made available to the user. The *getGPSLat* message generates the permissions {Access GPS, return Absolute Values, return relative values, return invalid values}. The last choice is important as it allows a user to bypass a process where a feature is blocked. Permissions are not implemented by the manufacturer and are therefore inaccessible. Messages which are executed contain a list of their associated permissions to track the types of generated data. In our example, the *getGPSLat* message has the "access GPS" permission associated with it. This autonomously generates a user interface entry in the user's dashboard, which is used for reporting. The user is free to change permission in real time, generating a superseding rule value which propagates between the environments. Each environment shares the latest version of a rule value and previous values are logged within telemetry and deleted.

E. REPORTING

A dynamic dashboard is built into the user's controlling application. This dashboard is generated by the framework, as driven by the manufacturer's messages and displays a set of user chosen data points. Continuing our example of the *getGPSLat* command, the dashboard will contain a pictograph of the number of message executions based on a user chosen duration. In this example, we could display the number of GPS coordinate queries, within a single day. Below the pictograph, a user can scroll through an enumerated list of return coordinates. This two fold process ensures a user knows when a GPS device is queried, and what data has

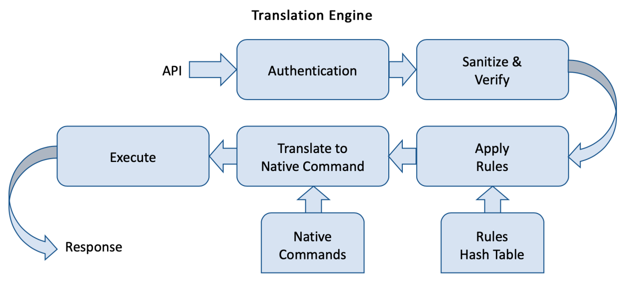

FIGURE 5. **Translation Flow.**

been returned. A user can subscribe to the available messages and pivot on the returned data as well as unsubscribe to dashboard entries which are no longer interesting. We provide a user the ability to customize the reporting tool and choose the duration and format of the report, a solution to the challenge of clearly reporting generated data. The framework contains built in templates to ease the customization process, where the templates represent a driven goal. For example, the default dashboard template aggregates all generated data which contains personally identifiable information. When a user registers a device with the manufacturer, the manufacturer creates a form for the user to fill out. This form generates a set of permissions, which generate dashboard entries. Entries which are either flagged by the manufacturer or the user as personally identifiable information, are included in the default template. The framework tracks its access and storage. Messages containing the registration data are reported to the dashboard as part of the default template. Blocking this information is as easy as setting the associated permission to deny, blocking all subsequent messages containing this data.

F. TRANSLATION LAYER

Bulwark's translation layer, shown in Fig. 5 is designed to authenticate, sanitize and verify a message. The rules engine extracts and applies each rule to the framework command. If allowed, The framework command is translated to a native IoT command and is executed on the device. The translation layer exists on both the cloud and the device and the code is nearly identical, with the exception of differences as represent within cloud dependencies. These dependencies are needed to assist with execution of cloud object commands. This pipeline uses a modular software paradigm, making it easy for a manufacturer to extend a device's command set, the framework to add rules and the development team to leverage revisions for deprecating and superseding an implementation.

1) AUTHENTICATION

A message's shared secret and signature are extracted, the message is unwrapped and the two security objects are verified. The signature and certificate are checked for validity and revocation, the shared secret is verified and the framework message is forwarded to the next step in the pipeline.

A framework message is passed in and the message is checked for lexical correctness. A comparison is made for each sub component of a message and an exception is created

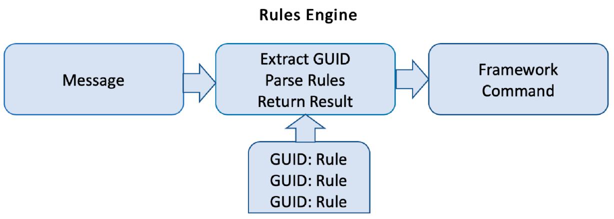

FIGURE 6. **Rule Validation.**

for any malformed data. The message is verified and extraneous data is stripped from the message as preparation for forwarding. If an exception exists, the process is halted and the exception is forwarded to the cloud service for logging. In the absence of an exception, the sanitized message is forwarded to the next step.

2) RULE TRANSLATION

A framework message is passed in and the message Globally Unique Identifier (GUID) is extracted. In this case, a GUID provides a method to uniquely identify a message. The existence of the GUID in the rules hash table is checked and an exception is created for its absence. A list of rules is extracted from the hash table as shown in Fig. [6] and each rule is parsed as a match for the current message. The logic of the rules engine follows a default deny scheme, where this value is overridden by the current rule result. If any rule resolves to deny, parsing ends and the process halts with an explicit deny. If none of the rules resolve to an explicit approve, the result is the same as before. Upon parsing completion if a deny rule is not matched and we at least one approve rule, the process continues to the next step.

3) COMMAND TRANSLATION

A framework message is passed in and the GUID is extracted. At this point we assume the passed in data is valid and we attempt to look up the message GUID in the command hash table as shown in Fig. [7]. If the table returns no data, the process ends without a result or an exception. The assumption is that the framework message is a stub and has yet to receive a corresponding native implementation. If a valid native command is returned, the command is executed within the IoT device and if applicable, a response is generated. Not all commands generate responses and all exceptions silently fail, with the exception of logging.

VI. EXPERIMENTAL RESULTS

The experiment process consists of either implementation or device simulation. We implement a device within our framework, or we simulated a devices' function within a Raspberry Pi. Our results include measuring the number of features, the complexity in implementation, the size of the resulting permission set and dashboard and the cost of hosting this device on a user's cloud account. We further define a feature as a function visible to the user and we define an internal feature as one or more functions, designed to support

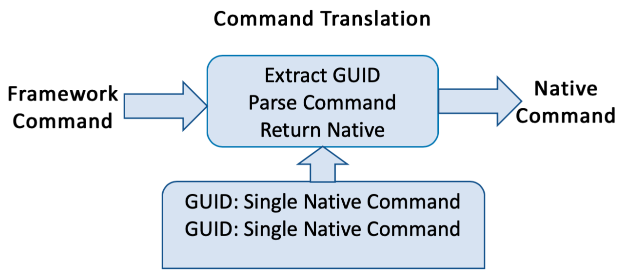

FIGURE 7. Command Translation.

a feature. This is the most difficult challenge within our effort as most IoT manufacturers lock down device implementation. For devices which supported an open source model, we were able to implement much of the device, however the remaining devices require either reverse engineering or simulation.

A. SMART PLUG

A smart Plug is a WIFI enabled electrical plug which uses a local application and a cloud service to provide the smart plug's state and user control. We Implemented the device, mobile application and cloud services within our framework and we simulated the home automation features. The framework produced seven permissions for six commands and a single permission to cover home automation extensions. Given additional work, the schema can support a granular breakdown of a feature's commands and produce a tree of permissions and messages. The cloud services cost for this device is within the free tier as network communication is on the order of kilobytes per hour. Storage is on the order of kilobytes and Lambda executions occur approximately once per minute. Each of the metrics fall within the free tier all three cloud services. The experimental data points are as follows.

- Feature - **Query device state**: {On, Off}. Encapsulates as a single message and contains a single permission. Perm: RemoteStatus {Allow, Deny, CIDR: Allow,Deny}. Returns the device state.
- Feature - **Toggle device state**: {On, Off}. Encapsulates as a single message and contains a single permission. Perm: RemoteStatus {Allow, Deny, CIDR: Allow,Deny}. Returns the device state.
- Feature - **Smart Home Access**: Set of native commands. Encapsulates any number of home automation native commands. Perm: AllowHomeAutomation {Allow, Deny, CIDR :Allow,Deny}. Native command response.
- Internal Feature - **Email registration for Remote Access**: Bypassed by the framework. The manufacturer currently requires a user to submit an email address, before enabling remote access and control. This feature is enabled by default and is controlled with the appropriate permissions.
- Internal Feature - **Store device State in the Cloud**: {On, Off}. Controls storing the device state as a database row and updating on request. Encapsulates as a single message and contains a single permission. Perm: StoreState {Allow, Deny, CIDR: Allow,Deny}. No response.
- Internal Feature - **Return GPS Data**: {On, Off}. Encapsulates as a single message and contains three permissions. Perm: EnableGPS {Allow, Deny}. Perm: AllowRemoteAccess {Allow, Deny, CIDR: Allow,Deny}. Perm: ReturnRelativeData {Allow, Deny, CIDR: Allow,Deny}. Return GPS Data based on permissions.

A summary of the smart plug is given in table [1].

TABLE 1. Smart Plug Summary.

SMART PLUG				
Feature	Description	State	Rule	Cloud Impact
Query State	Request state	{On,Off}	Disables request by network	Within free tier
Toggle State	Set state	{On,Off}	Disables reques t by network	Within free tier
Smart Home Access	Allow device access to third part APIs	{Enable,Disable}	Disable access by network	No impact
Email Registration	Blocks email address requirement	{On,Off}	Disable address requirement	No Impact
Store State in the Cloud	Remote storage of current state	{On,Off}	Disable remote storage of the state	Within free tier
Access GPS	Access current device position	{On,Off,Networ k,Fake Data}	Disables by network or real data	Within free tier

TABLE 2. Smart Scale Summary.

SMART SCALE				
Feature	Description	State	Rule	Cloud Impact
Query Data	Request user data	{Enable,Disable}	Controls data access	No impact
Toggle Cloud Data	Request remote data	{Enable,Disable}	Controls data access	No impact
Remote Delete	Controls delete access	{Enable,Disable}	Controls deletion	No impact
Third Party Integration	Controls third party data access	{Allow,Deny}	Control data access	No Impact

B. SMART SCALE

A smart scale is a Wi-Fi enabled body scale designed to capture human weight and store the results in a mobile application and a cloud service. The product supports multiple users, but requires the registration of an email account for each. Lack of registration limits the function a local display of weight. The device generates a timestamp, a weight and correlates the data to a user. The data is shared with a mobile application and stored in a cloud account associated with the manufacturer. The device ships with a mobile application, but supports the more common fitness applications alluding to a common communication language. Both the scale and the associated mobile application store previous data, while the scale supports an API call to return stored data to the application. In Bulwark, we treat historical data the same as

TABLE 3. **Garage Door Opener Summary.**

Garage Door Controller				
Feature	Description	State	Rule	Cloud Impact
Query Door State	Request state	{Enable,Disable}	Controls state access by network	No impact
Toggle Door State	Set state	{Enable,Disable}	Control door position	No impact
Store Door State in the Cloud	Request state via remote network	{Enable,Disable, network}	Query state via the cloud API	No impact

current data, in that all data contains a time stamp, a weight and a user. We de-duplicate responses by storing data as a database row, keyed off of the user and the timestamp. The cloud database is treated as the authority for user data and deletion requests require a follow up user data request to the cloud, which updates the mobile application.

- Feature - **Query User Data**: A query request, blocking on device response. Encapsulates as a single message and contains two permissions. Perm: AllowUserQuery {Allow, Deny} and Perm: AllowUserQuery(User) {Allow, Deny}. Returns historical user data null.
- Feature - **Query Cloud User Data**: A query request, blocking on device response. Encapsulates as a single message and contains two permissions. Perm: AllowUserQuery {Allow, Deny} and Perm: AllowUser-Query(User) {Allow, Deny}. Returns historical user data.
- Feature - **Remote Delete**: A request to delete an instance of data. Currently supported in the cloud, but not the device. Encapsulates as a single message and contains a single permission. Perm: AllowRemoteDeletion {Allow, Deny}.
- Feature - **Allow Third Party Integration**: A boolean value to set third party support. Encapsulates as a single message and contains a single permission. Perm: AllowThirdParty (Optional third party parameter) {Allow, Deny}.

A summary of the smart scale is given in table [2].

C. OPEN SOURCE GARAGE DOOR CONTROLLER

A smart garage door controller is a Wi-Fi enabled home automation device which is designed for third party home automation solutions. This device contains an open source solution which supports determining door position and movement of a user's garage door. The device accepts commands to open and close a garage door and returns the current position. The open source nature of the software and the supplied SDK allows us to easily integrate this device into Bulwark. The device supports a number of third party solutions as well as programming language clients.

- Feature - **Query Door State**: A query request, blocking on device response. Encapsulates as a single message and contains two permissions. Perm: AllowDoorState {Allow, Deny} and Perm: AllowDoorStateRemote {Allow, Deny CIDR: Allow, Deny}.

TABLE 4. **Smart Switch Summary.**

Closed Source Smart Switch				
Feature	Description	State	Rule	Cloud Impact
Query Switch State	Request State	{Enable,Disable}	Controls access by network	No impact
Toggle Switch State	Set State	{Enable,Disable}	Controls switch by network	No impact
Set Dimmer State	Set State	{Enable{int},Disable}	Controls dimmer by network	No impact
Toggle Switch State Using Rules	Set State with Rules	{Enable,Disable} API:{Enable,Disable}	Controls rules processing and variable definition	No Impact

TABLE 5. **Device Summary.**

Summary of Results			
Device	Feature	Rules	Impact
Smart Plug	Six	Five	None
Smart Scale	Seven	Nine	Expanded Feature Set
Garage Door Controller	Three	Four	Difficult to Integrate
Smart Switch	Four	Six	Expanded Feature Set

- Feature - **Toggle Door State**: Activate garage door opener. Encapsulates as a single message and contains two permissions. Perm: ToggleDoor {Allow, Deny} and Perm: ToggleDoorRemote {Allow, Deny CIDR: Allow, Deny}.
- Feature - **Store Door State in the Cloud**: A Boolean value settings the remote storage option for the door state. Encapsulates as a single message and contains a single permission. Perm: StoreStateCloud {Allow, Deny}.

A summary of the smart garage door opener is given in table [3].

D. CLOSED SOURCE SMART SWITCH

A smart switch is a Wi-Fi enabled electrical light switch, designed for third party home automation solutions. The switch is part of a larger home automation solution, designed to interact with a closed source mobile application. The switch contains firmware which is modified via flashing and the process is automated by periodic remote checks for a new version of firmware. The switch hardware incorporates on-off functionality and a five step dimmer switch, which is designed to control any number of light bulbs. The hardware constrains the supported devices to a limit of 300 watts and does not support any item of electrical motor. It was a challenge to incorporate this device into our framework, as the work required reverse engineering the on-off and dimmer function commands. We were able to successfully control the switch, albeit with intermittent success. Utilizing the two control states, we were able to implement the commercial features and expand on the feature set by including our own design. The new feature includes a rules based engine, which is designed to support simple variables. Variables are defined with simple assignment or an API call. In the case of an API call, a timer is set to query an API and set the variable to the returned value. This case supports dynamic function such as setting $sunrise and $sunset to the current day's values. We can then set a rule to enable the switch at $sunset and disable at $sunrise.

- Feature - **Query Switch State**: A query request, blocking on device response. Encapsulates as a single message and contains two permissions. Perm: AllowSwitchState {Allow, Deny} and Perm: AllowSwitchStateRemote {Allow, Deny CIDR: Allow, Deny}.
- Feature - **Toggle Switch State**: Activate lights via the switch control. Encapsulates as a single message and contains two permissions. Perm: ToggleSwitch {Allow, Deny} and Perm: ToggleSwitchRemote {Allow, Deny CIDR: Allow, Deny}.
- Feature - **Set Dimmer Switch State**: Activate lights using one of the five dimmer states. Encapsulates as a single message and contains two permissions. Perm: SetSwitchint {Allow, Deny} and Perm: SetSwitchRemoteint {Allow, Deny CIDR: Allow, Deny}.
- New Feature - **Toggle Switch State Based on Rules**: Dependent upon the rules for toggle switch and dimmer settings. Perm: ProcessRules{Allow, Deny CIDR: Allow, Deny} and Perm: DefineByAPI{Allow, Deny CIDR: Allow, Deny}

A summary of the smart switch is given in table [4].

E. OVERHEAD

In it's current form there is little overhead to account for when using our framework. Cloud manufacturers offer free accounts, with generous limits on free tier usage. Where applicable, the rules creation and translation process uses a negligible amount of additional battery and storage. The cloud component of the framework can be constrained to the free tier and the overall network overhead is less than 10%, when compared to a device implementation sans framework. Overhead of our proposed framework can be summarized as follows.

- Single digit percentage increase in traffic size.
- A small increase in the number of simultaneous connections as we introduce as greater coupling between the device, the cloud and the mobile application.
- Storage growth is minimal as we remove duplicate messages and older message versions.

In summary, the growth in storage and network communication is offset by the security and privacy features built into the product. A summary of device results is given in table [5]

VII. DISCUSSION

Convincing users at scale to host their own cloud service and manufacturers to compromise on data access is a significant challenge to our success. Although some users may see the immediate benefit to self hosting their device's cloud services, most people are not adept in software security and would consider the change to be overly complex. Conversely, device manufacturers will favor the current system as it favors monetizing user data. Despite having a high bar of entry, we believe inroads can be made by emphasizing the positive attributes of a privacy minded framework. The key to success is the combination of making it easier to acquire and use cloud credentials and a framework which reduces a manufacturers cost to implement and maintain a myriad of devices.

A. USER PERSPECTIVE

Advantages of the proposed framework from a user's perspective are as follows.

- **Ease of Use**: The mobile application enables a user to set up a cloud services solution using their own credentials. Once setup is complete the framework doesn't require additional maintenance to operate a device.
- **Scalability**: The framework supports any number of IoT devices with a single application and instance of cloud services.
- **Data Transparency**: The mobile application contains a telemetry panel, which can be customized for tailored data reporting. We support the ability to render data generation by message and location as well as by data propagation and storage location. The cloud service is available for inspection and the user can log in to parse their data.
- **Secure Data Storage**: A user's data is secured in the cloud, using their own credentials. The manufacturer does not have access to the data and the user has sole discretion in the data life-cycle.
- **Enforceable Rules**: The user is presented with a set of rules per device message. the rules govern the scope and nature of message behavior and the user can modify the rules at will.

B. DEVICE MANUFACTURER PERSPECTIVE

Advantages of the proposed framework from a device manufacturer's perspective are as follows.

- **Manufacturing Costs**: The fixed cost of implementing our framework works well within the forecasting process of determining the overall manufacturing cost. It's easy to determine engineering cost when design and implementation can be sized correctly for a project.
- The modular approach to the framework allows a manufacturer to implement and support multiple devices, using a single suite of software. When engineering overlap is found between devices, our framework makes it easy to plugin previous solutions, reducing the overall cost of a project.
- Development overlap is used to further reduce costs. Once a development team successfully implements a device, the cost of supporting additional devices is reduced. As a developer becomes adept in using the IDE, more of the work will involve the core implementation and less of the time will be spent solving ancillary problems.

- **Ease in Manufacturing**: The automation process works well with a factory model where manufactures can choose to create dynamic manufacturing lines, E.G. lines which switch seamlessly between devices, without having to adjust the software process.
- Our approach can lead to shorter device manufacturing times, less bugs, which leads to fewer support resources post shipping, and a positive experience by working in a single ecosystem.

VIII. FRAMEWORK EXTENSIONS

We are exploring the use of private cloud services, credential auto-generation and pooling, and the use of a single cloud service which would securely support multiple users.

A. PRIVATE CLOUD

The private cloud is an instance of cloud services executing in a customers local environment and each of the cloud service providers offer a private cloud solution. Leveraging the advantages of the framework and the private cloud to include the following.

- **Encapsulation of Network Traffic**: A private cloud enables network traffic containment within a customer's secure environment.
- **Control of Cloud Services**: Executing an instance of a private cloud enables a customer to retain ownership of cloud administration. Removing the attack vector of a publicly accessible cloud, facilitates the hardening of the cloud service's security.
- **Greater Control of Monitoring**: Although a public cloud offers a diverse set of environment monitoring, a private cloud enhances this ability. Given the location of a private cloud, a customer can lock down network routers and further control cloud access. Telemetry can be deployed to the private cloud servers, providing data outside of the virtual environment.

B. SECURE COALESCING OF CLOUD SERVICES

A cloud service is by design a set of virtual objects which are designed to provide function at an extraordinary scale. The design of our framework calls for a per user credential deployment of a cloud service environment to ensure the separation of user data from the manufacturer and from other users. This aspect of the design is a compromise of our original design, where we worked to derive a brokered solution. A abstracted brokering form would accept messaging from all customers, based on the supported IoT device and then securely store data in a way that prevents data leakage. Further research into a provably secure broker would include.

- **A Single API Per Collection of Devices**: Given a set of supported IoT devices, an ideal solution would accept all network communication from the devices and securely process the result.
- **Centralized Database**: A secure solution would allow for the storage of all customer data into a single set of databases and storage buckets, using a schema which prevents any one user from accessing another user's data. This includes the prevention of a manufactured ''back door'' giving customer data access to the manufacturer.
- **Data Encryption and Authentication Schema**: By utilizing a process which signs and encrypts user data, we mathematically prove data is hidden from unauthorized users and the data has not been tampered with.

C. SECURE ACCESS ACROSS CREDENTIAL BOUNDARIES

Part of the unfinished design for this framework includes a feature which allow users to broker temporary permission to other user accounts. Given that much of this framework's design involves the enforcement of permissions, it is a natural fit to include the ability to extrapolate permissions across user accounts. For example, user one could allow user two to access his webcam, but only for a short period of time. At the conclusion of the duration, we could provably revoke access and restore the original permission set. An ideal design for this feature would be the following.

- **Cross Account Permissions**: A schema which supports the ability to grant another user access to your own data. The schema requires the ability to describe a delineated pool of data, access type and the ability to set a duration, revoking on expiration. Cross account permissions are stored in the same permissions table as local permissions, and function in the exact same manner. The only difference between the permission's type is an expansion of meta-data, to support cross credential and transitional responses.
- **Transitional Encryption Schema**: An ideal solution would have a default behavior of encrypting a user's data. When a cross account permission is set, the affected data is decrypted and encrypted using the new user's account. The certificate used to encrypt and sign the data contains an expiry data which coincides with the permission expiration. DRM is used to prevent client side copying of the data and the data is destroyed upon expiration.
- **Brokering Process**: The last component of this design, is an engine which processes cross credential permissions and schedules decryption, storage, encryption and destruction of data jobs. The process executes in real-time, using encryption and certificates to enforce job result. A job's response contains the execution result and job meta-data, which is used to support post execution logging and a reference to the acted upon data. The result is logged as a database entry and used for postmortem analyses.

IX. CONCLUSION

The proliferation of IoT devices raises serious concerns for a user's privacy and security. Features such as video and audio capture, real time location data and remote processing are normal industry practices. The expectation of growth in this area lends a sense of urgency to finding a balance between feature rich products and a user's privacy. We propose building a unified solution into a development framework to provide the needed balance between product revenue and privacy. Using a smart plug and garage door controller, we incorporated a basic implementation into the framework, enabling secure interaction over remote networks and stored the device state and an historical record of interaction in a user's cloud account. We autonomously generated rules for each message, enforcing device behavior and bypassed the manufacturer's cloud account. The smart scale and switch required reverse engineering and simulation of behavior to function within our framework. We created an implementation which provided basic interaction with the device and then extended this function to features not provided by the manufacturer. The smart scale stored a user's weight in the cloud and used dashboards to render a user's result over a selected period. We expanded on the switch's function such that a user can create executable variables using local assignment and remote APIs. We created a pair of variables to represent sunrise and sunset, assigned date-time values from a remote website and created a pair of rules to toggle the switch state based on the position of the sun. Proposed framework can be incorporated to include other IoT devices. Finally, overhead was negligible as compared to a non framework implementation. Network usage grew by less than 10% while storage grew less than 2%. Devices with open source solutions execute our framework with no impact to performance and network latency did not increase, while simulation rendered a small growth in network and memory usage.

REFERENCES

[1] *Analyzing Research Trends*. Accessed: Oct. 1, 2021. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0167404817300603

[2] *Us Troops Accidentally Reveal Secret Bases*. Accessed: Oct. 1, 2021. [Online]. Available: https://www.popularmechanics.com/technology/apps/a15912407/strava-app-military-bases-fitbit-jogging/

[3] *Cisco Internet of Things*. Accessed: Oct. 1, 2021. [Online]. Available: https://www.cisco.com/c/dam/en/us/products/collateral/se/internet-of-things/at-a-glance-c45-731471.pdf

[4] *IoT Market Predicted to Double, Reaching $ 520b*. Accessed: Oct. 1, 2021. [Online]. Available: https://www.forbes.com/sites/louiscolumbus/2018/08/16/iot-market-predicted-to-double-by-2021-reaching-520b/#82674f91f948

[5] *IoT Security Spending to Reach $ 1.5 Billion*. Accessed: Oct. 1, 2021. [Online]. Available: https://www.zdnet.com/article/iot-security-spending-to-reach-1-5-billion-in-2018

[6] J. Hong, A. Levy, L. Riliskis, and P. Levis, "Don't talk unless i say so! securing the Internet of Things with default-off networking," in *Proc. IEEE/ACM 3rd Int. Conf. Internet Things Design Implement. (IoTDI)*, Oct. 2018, pp. 117–128.

[7] K. E. Benson, Q. Han, K. Kim, P. Nguyen, and N. Venkatasubramanian, "Resilient overlays for IoT-based community infrastructure communications," in *Proc. IEEE 1st Int. Conf. Internet Things Design Implement. (IoTDI)*, Apr. 2016, pp. 152–163.

[8] M. Chung, "One-to-Many data transmission for smart devices at close range," in *Proc. IEEE 1st Int. Conf. Internet Things Design Implement. (IoTDI)*, Apr. 2016, pp. 265–270.

[9] J. Hall and R. Iqbal, "CoMPES: A command messaging service for IoT policy enforcement in a heterogeneous network," in *Proc. IEEE/ACM 2nd Int. Conf. Internet Things Design Implement. (IoTDI)*, Apr. 2017, pp. 37–44.

[10] M. H. Mazhar and Z. Shafiq, "Characterizing smart home IoT traffic in the wild," in *Proc. IEEE/ACM 5th Int. Conf. Internet Things Design Implement. (IoTDI)*, Apr. 2020, pp. 203–215.

[11] H. Nguyen, R. Ivanov, L. T. X. Phan, O. Sokolsky, J. Weimer, and I. Lee, "LogSafe: Secure and scalable data logger for IoT devices," in *Proc. IEEE/ACM 3rd Int. Conf. Internet Things Design Implement. (IoTDI)*, Oct. 2018, pp. 141–152.

[12] P. Bovornkeeratiroj, S. Iyengar, S. Lee, D. Irwin, and P. Shenoy, "RepEL: A utility-preserving privacy system for IoT-based energy meters," in *Proc. IEEE/ACM 5th Int. Conf. Internet Things Design Implement. (IoTDI)*, Apr. 2020, pp. 79–91.

[13] H. Kim, E. Kang, E. A. Lee, and D. Broman, "A toolkit for construction of authorization service infrastructure for the Internet of Things," in *Proc. 2nd Int. Conf. Internet Things Design Implement.*, Apr. 2017, pp. 147–158.

[14] X. Li, H. Wang, Y. Yu, and C. Qian, "An IoT data communication framework for authenticity and integrity," in *Proc. 2nd Int. Conf. Internet Things Design Implement.*, Apr. 2017, pp. 159–170.

[15] A. Rullo, D. Midi, E. Serra, and E. Bertino, "A game of things: Strategic allocation of security resources for IoT," in *Proc. 2nd Int. Conf. Internet Things Design Implement.*, Apr. 2017, pp. 185–190.

[16] H. Chiang, J. Hong, K. Kiningham, L. Riliskis, P. Levis, and M. Horowitz, "Tethys: Collecting sensor data without infrastracture or trust," in *Proc. IEEE/ACM 3rd Int. Conf. Internet Things Design Implement. (IoTDI)*, Apr. 2018, pp. 249–254.

[17] F. De Vita, D. Bruneo, and S. K. Das, "A novel data collection framework for telemetry and anomaly detection in industrial IoT systems," in *Proc. IEEE/ACM 5th Int. Conf. Internet Things Design Implement. (IoTDI)*, Apr. 2020, pp. 245–251.

[18] I. Pasquel Mohottige, H. H. Gharakheili, A. Vishwanath, S. S. Kanhere, and V. Sivaraman, "Evaluating emergency evacuation events using building WiFi data," in *Proc. IEEE/ACM 5th Int. Conf. Internet Things Design Implement. (IoTDI)*, Apr. 2020, pp. 116–127.

[19] F. Renna, J. Doyle, V. Giotsas, and Y. Andreopoulos, "Query processing for the Internet-of-Things: Coupling of device energy consumption and cloud infrastructure billing," in *Proc. IEEE 1st Int. Conf. Internet Things Design Implement. (IoTDI)*, Oct. 2016, pp. 83–94.

[20] W. Wang, J. Su, Z. Hicks, and B. Campbell, "The standby energy of smart devices: Problems, progress, potential," in *Proc. IEEE/ACM 5th Int. Conf. Internet Things Design Implement. (IoTDI)*, Oct. 2020, pp. 164–175.

[21] M. Haus, J. Ott, and A. Y. Ding, "DevLoc: Seamless device association using light bulb networks for indoor iot environments," in *Proc. IEEE/ACM 5th Int. Conf. Internet-of-Things Design Implement. (IoTDI)*, May 2020, pp. 231–237.

[22] G. Vaidya, A. Nambi, T. V. Prabhakar, V. Kumar T, and S. Sudhakara, "IoT-ID: A novel device-specific identifier based on unique hardware fingerprints," in *Proc. IEEE/ACM 5th Int. Conf. Internet Things Design Implement. (IoTDI)*, Apr. 2020, pp. 189–202.

[23] C. Ruiz, S. Pan, A. Bannis, M.-P. Chang, H. Y. Noh, and P. Zhang, "IDIoT: Towards ubiquitous identification of IoT devices through visual and inertial orientation matching during human activity," in *Proc. IEEE/ACM 5th Int. Conf. Internet Things Design Implement. (IoTDI)*, Apr. 2020, pp. 40–52.

[24] S. S. Sandha, J. Noor, F. M. Anwar, and M. Srivastava, "Time awareness in deep learning-based multimodal fusion across smartphone platforms," in *Proc. IEEE/ACM 5th Int. Conf. Internet Things Design Implement. (IoTDI)*, Oct. 2020, pp. 149–156.

[25] D. Hu and B. Krishnamachari, "Fast and accurate streaming CNN inference via communication compression on the edge," in *Proc. IEEE/ACM 5th Int. Conf. Internet Things Design Implement. (IoTDI)*, Apr. 2020, pp. 157–163.

[26] C. Wang, C. Gill, and C. Lu, "Adaptive data replication in real-time reliable edge computing for Internet of Things," in *Proc. IEEE/ACM 5th Int. Conf. Internet Things Design Implement. (IoTDI)*, Apr. 2020, pp. 128–134.

[27] V. Cozzolino, J. Ott, A. Y. Ding, and R. Mortier, "ECCO: Edge-cloud chaining and orchestration framework for road context assessment," in *Proc. IEEE/ACM 5th Int. Conf. Internet Things Design Implement. (IoTDI)*, Jul. 2020, pp. 223–230.

[28] M. Norris, B. Celik, P. Venkatesh, S. Zhao, P. McDaniel, A. Sivasubramaniam, and G. Tan, "IoTRepair: Systematically addressing device faults in commodity IoT," in *Proc. IEEE/ACM 5th Int. Conf. Internet Things Design Implement. (IoTDI)*, Apr. 2020, pp. 142–148.

[29] Y. Gao, J. Zhang, G. Guan, and W. Dong, "LinkLab: A scalable and heterogeneous testbed for remotely developing and experimenting iot applications," in *Proc. IEEE/ACM 5th Int. Conf. Internet Things Design Implement. (IoTDI)*, Jan. 2020, pp. 176–188.

[30] T. Wolf and A. Nagurney, "A layered protocol architecture for scalable innovation and identification of network economic synergies in the Internet of Things," in *Proc. IEEE 1st Int. Conf. Internet Things Design Implement. (IoTDI)*, Apr. 2016, pp. 141–151.

[31] J. Venkatesh, C. Chan, A. S. Akyurek, and T. S. Rosing, "A modular approach to context-aware iot applications," in *Proc. IEEE 1st Int. Conf. Internet Things Design Implement. (IoTDI)*, Feb. 2016, pp. 235–240.

[32] S. Qanbari, S. Pezeshki, R. Raisi, S. Mahdizadeh, R. Rahimzadeh, N. Behinaein, F. Mahmoudi, S. Ayoubzadeh, P. Fazlali, K. Roshani, A. Yaghini, M. Amiri, A. Farivarmoheb, A. Zamani, and S. Dustdar, "IoT design patterns: Computational constructs to design, build and engineer edge applications," in *Proc. IEEE 1st Int. Conf. Internet-of-Things Design Implement. (IoTDI)*, Apr. 2016, pp. 277–282.

[33] *Utilizing Vehicle Data in Law Enforcement Investigations*. Accessed: Oct. 1, 2021. [Online]. Available: https://www.iacpcybercenter.org/wp-content/uploads/2020/09/Vehicle-Data_LECC-Article.pdf

[34] *12 Days of Vehicle Forensics*. Accessed: Oct. 1, 2021. [Online]. Available: https://berla.co/12-days-of-vehicle-forensics/

[35] *MSAB Raven*. Accessed: Oct. 1, 2021. [Online]. Available: https://www.msab.com

[36] *Terraform*. Accessed: Oct. 1, 2021. [Online]. Available: https://www.terraform.io/

[37] *AWS Lambda@edge*. Accessed: Oct. 1, 2021. [Online]. Available: https://aws.amazon.com/lambda/edge/

[38] *AWS Lambda*. Accessed: Oct. 1, 2021. [Online]. Available: https://aws.amazon.com/lambda/

[39] *Amazon Elasticache*. Accessed: Oct. 1, 2021. [Online]. Available: https://aws.amazon.com/elasticache/

[40] *Amazon S3*. Accessed: Oct. 1, 2021. [Online]. Available: https://aws.amazon.com/s3/

JEREMY LYNN REED received the B.S. and M.S. degrees in computer science from The University of Texas at San Antonio, San Antonio, TX, USA, in 2007 and 2016, respectively, where he is currently pursuing the Ph.D. degree.

He spent eight and half years at Microsoft as a Software Security Engineer, developing and securing web technologies. He is currently a Performance and Security Engineer at DocuSign. His research interests include software security with an emphasis in web-based technology and IoT performance and security.

ALI ŞAMAN TOSUN received the B.S. degree in computer engineering from Bilkent University, Ankara, Turkey, in 1995, and the M.S. and Ph.D. degrees from The Ohio State University, in 1998 and 2003, respectively.

He worked at the Department of Computer Science, The University of Texas at San Antonio, from 2003 to 2021. He is currently the Allen C. Meadors Endowed Chair of computer science at The University of North Carolina at Pembroke, Pembroke, NC, USA. His research interests include software-defined networking, network security, storage systems, and large-scale data management.