# Just the FACTS: Flexible and Energy Efficient Federated Access Control for the Edge

Shereen Elsayed, Chandra Krintz, Rich Wolski
Department of Computer Science
University of California, Santa Barbara
Santa Barbara, USA 93106
{s_elsayed,ckrintz,wolski}@ucsb.edu

***Abstract*—**
**We present Federated Access ConTrol Support (FACTS) for flexible and energy efficient IoT edge deployments. FACTS is a hybrid approach to access control that couples capability support over MQTT with TLS based security to reduce the power consumption of IoT deployments. Capability systems are lightweight and enable fine-grained access control using cryptographically secure tokens. TLS provides number of security guarantees but is expensive in terms of both time and energy when employed by resource constrained devices at the edge. By combining the two, FACTS enables IoT deployment administrators to decide where and when to use encrypted communication for access control to achieve the best trade-off between data privacy, performance, and energy efficiency. We evaluate FACTS for different deployment scenarios and show that it improves both energy use and performance relative to existing approaches.**

***Index Terms*—cloud continuum, TLS, capability-based security, hybrid access control, federation**

## I. Introduction

The *Internet of Things* (IoT) is a technological vision in which people interact with their environment (an environment populated by "things") digitally, using ubiquitous network connectivity to facilitate the interactions. IoT has great potential for enhancing situational awareness and performing automated and intelligent actuation and control of cyber-physical systems.

However, the growth and adoption of this compelling vision has been slow. IoT deployments are very challenging to program, debug, secure, and maintain. End-to-end, an IoT application must be able to marshal a vastly heterogeneous collection of compute, storage, and networking elements using an amalgamation of software technologies often not designed to work as an ensemble. Further, many of the most compelling use cases from climate change, disaster prediction and response, ecology, etc. require deployments that are remote, inaccessible, hostile to electronics, that lack power infrastructure (forcing reliance on batteries), and that must span trust domains (edge versus cloud).

Most recently, IoT technologies have increasingly focused on the use of the cloud as an "always on," ubiquitously accessible, and flexible digital infrastructure. Connecting every "thing" to the cloud provides a way to tame the heterogeneity inherent in the paradigm. Once a "thing" is interfaced to a cloud, all of the scalable, secure, and performant services hosted by that cloud are available to an IoT application designed to use that "thing."

While most, if not all, of the public cloud venues provide support for IoT development and integration [1], [2], perhaps the most popular technologies belong to Amazon Web Services (AWS). AWS IoT [3] enables centralized management of IoT deployments that consist of sensors and resource-constrained devices (called *things*). Things can communicate and interact directly with AWS cloud services when they are Internet-connected.

However, to reduce latency and support functionality during periodic or temporary interruptions of network connectivity to and from the cloud, AWS also supports a remote interfacing technology called *Greengrass* [4]. Greengrass is designed to deploy to an edge device (typically a single board computer with Java support). It provides services directly to things or acts as a proxy for AWS cloud services.

The edge device that executes the Greengrass service is termed a Greengrass (GG) *core* device. It provides authentication, authorization, an MQTT [5] broker (implementing a popular publish-subscribe communication protocol for IoT), as well as computational components (e.g. AWS Lambda functions [6]) for things to use as services in an IoT deployment. As such, GG support enables low latency response and data-driven intelligence (via local, edge-based ML inference and user-defined data analysis components) for things. It also enables the "off-line" operation of IoT deployments when temporarily disconnected from the cloud. Finally, by supporting MQTT communication, this combination of services enables a wide range of heterogeneous devices to be integrated as things, including those connected to the core via popular IoT/sensor network protocols (e.g. zigbee [7], [8], z-wave [9], [10], LoRaWAN [11], [12]).

Unfortunately, while IoT technologies such as Greengrass are effective with respect to enabling the cloud to act as infrastructure for IoT, they are not designed to provide the extreme energy efficiency required for remote, battery powered, edge devices and things. For example, a GG core's components must execute continuously to implement its operation. Although the core can be disconnected from the cloud intermittently, it requires connectivity for setup and device commissioning (thing configuration, authentication, and integration). Moreover, AWS IoT and GG core uses Transport Layer Security (TLS) [13] protocols, based on public-key cryptography [14] to authenticate and authorize all thing interactions. It also uses TLS to provide an encrypted channel for thing data transmission (over MQTT). TLS is complex to configure correctly and consumes significant resources (and thus battery power) on all devices in an IoT deployment. It cannot be disabled since it is the basis for the AWS security model. Thus IoT applications where the perfect-forward-secrecy [15] that TLS implements is not needed (e.g. devices connected to a secure network behind a firewall) must nonetheless expend the energy necessary to support it. Similarly, many applications require only secure authentication and/or tamper-proofness, not a fully private communication channel (e.g. temperature data from a farm, car counts from a bridge, anonymized data, etc.)

To address these limitations, we present FACTS – **F**ederated **A**ccess **C**on**T**rol **S**upport for flexible and energy efficient authentication and authorization, for remote, battery powered IoT edge deployments. FACTS is a hybrid approach to access control that combines capability-based authentication with TLS based AWS IoT/GG core security to reduce the power consumption of edge devices. Capability systems enable fine-grained access control using cryptographically secure tokens [16]–[18]. FACTS extends capability-based access control with support for MQTT communication and integration with the GG core to enable end-to-end authentication and authorization with significantly lower energy consumption. FACTS also enables IoT deployment administrators to decide where and when to use encrypted communication (in addition to authentication and authorization) to achieve the best trade-off between data privacy, performance, and energy use.

We evaluate FACTS using small, single board computers that act both as core devices (i.e. those hosting a GG core) and as representatives of sensors and actuators. We make this latter choice (as opposed to using embedded devices for the sensors and actuators) because it is possible to achieve more accurate instrumentation of an edge-based IoT deployment end-to-end. We measure energy consumption using a logic analyzer and digital circuit that facilitates fine-grained, accurate sampling of current and voltage. Our results show that the combination of capability-based authentication and TLS to facilitate communication between the GG core and AWS dramatically reduces energy consumption at the edge. On average at the core device, FACTS reduces energy consumption by 11% when client devices that employ a duty cycle. It reduces energy consumption of the core by up to 95% when client devices send multiple messages and it reduces client-side energy consumption (by up to 74%).

## II. Related Work

The heterogeneity and dynamic and distributed nature of IoT deployments make resource access control a significant challenge. Many commercial IoT systems use Transport Layer Security (TLS) [13] protocols, based on public-key cryptography [1], [3]. These systems use edge devices to communicate with servers and edge proxies via encrypted channels, often using RSA certificates. Although TLS addresses many security challenges, it also consumes significant resources on resource restricted devices (memory, computation, network, battery power, etc.) and depends on a number of resource-intensive operations for its security.

Due to the implementation complexity, resource consumption, and configuration difficulty of TLS-based security, many IoT devices have weak or no access control, and are easily compromised [19], [20]. As a result, many devices are placed behind gateways or firewalls which proxy requests [21], rendering the resource expense associated with TLS-based security on the device needlessly redundant and costly. Related work, e.g. [14], [22], shows that it is possible to reduce the overhead of TLS significantly in some edge computing contexts. Our work reduces the overhead of TLS used by cloud-IoT systems by integrating capability based security. Capability systems provide flexible, fine-grained control access and data integrity without the use of expensive encryption, identity, and certificate management [16], [17], [23].

Other prior works also use hybrid approaches to access control to achieve combined benefits (security and performance) for IoT deployments [24]. Pranata et al in [25] use public key encryption technologies with capabilities in IoT settings to reduce resource use. In [26], the authors combine role based, attribute based, and capability based access control for IoT deployments. Our approach extends TLS by embedding capabilities into the MQTT payload from devices. The authors in [27] embed attribute based access control but require enforcement monitoring. MQTT can also use TLS-based access control directly, but doing so for IoT is heavy weight and vulnerable to attack [28]. Finally, new standards are emerging to reduced the overhead of security protocols on resource constrained devices [29], [30].

## III. FACTS

FACTS is a lightweight augmentation of TLS-based security mechanisms provided by public cloud systems that enables more flexible and energy-efficient authentication and authorization services (henceforth termed simply as *Auth Services or AS*) for fog and edge devices. To enable these benefits, we design FACTS around these key security principles.

- AS must implement access control based on verified identities.
- AS must integrate support for auditing access and enforcing accountability for actions taken.
- AS policies must be able to specify permissions at a granular level (e.g. for individual operations, resource, and data types).
- AS must be able to adapt to dynamically changing deployment conditions and configurations.
- Security requirements can be specified at a granular level (e.g. authentication, privacy, integrity, access control).
- Edge deployments must be able to make progress (i.e. continue secure execution) when disconnected from the cloud for extended periods.
- AS must be able to tolerate faults and work consistently and efficiently under a wide range of resource constraints.

The first two principles are standard for AS. The others are tailored to the characteristics of remote IoT deployments. Given that these deployments are heterogeneous with a wide range of capabilities, capacities, and resource restrictions, security cannot be "one size fits all". In particular, TLS addresses many security requirements but many requirements may be redundant or excessive (e.g. when behind a firewall or when authentication and data integrity and not full data privacy is sufficient). Given that TLS also consumes significant resources (memory, computation, network, battery power, etc.) and depends on a number of resource-intensive operations for its security, we design FACTS to support a range of TLS and non-TLS access controls that are granular and that can be tailored to the security requirements of individual deployments.

Such flexibility should also extend to the IoT application runtime. Thus, our principles also target dynamic adaptation of access control policies to support changing deployment conditions and requirements. Finally, an AS must be designed such that an application can continue to securely operate (and make progress) in the face of faults, intermittent connectivity, and lack of cloud access given that such scenarios are common for many remote IoT deployments.

### A. *Design*

To support these principles, FACTS augments TLS-based AS with fine-grained access control. For this to work in remote, failure-prone IoT settings, we design FACTS with the following features: (1) edge-only device commissioning and authorization, (2) minimal bespoke client software hosting, (3) easy integration with existing public cloud edge deployments,(4) fault resiliency, (5) logging, and (6) end-to-end energy efficiency. Our goal is to provide a range of security guarantees that can be used when full-TLS is not required in an attempt to reduce the energy use of battery-powered edge devices for AS. For (1), we eliminate reliance on the Internet and cloud connectivity, thus ensuring that IoT deployments can maintain their security integrity even when disconnected from the cloud (i.e. when they are only locally connected). Because of the vast heterogeneity of client devices in IoT deployments, we designed FACTS (2) so that it can work on a large majority of devices without modification. To enable this, we layer FACTS on top of MQTT – a popular publish-subscribe protocol.

At the other end of the IoT spectrum, our goal is to make it easy to integrate FACTS with public cloud IoT edge services (3), e.g. AWS Greengrass or Azure IoT Hub, thereby facilitating its use by existing IoT deployments. In particular, FACTS must be able to provide the same or similar functionality as these services for client devices, to be useful. FACTS features (4) and (5) are key to addressing the aforementioned design principles for IoT (auditing and fault resiliency.) Finally, FACTS customizes access control to trade off some TLS guarantees for energy savings; thus our approach must achieve significant energy savings (6).

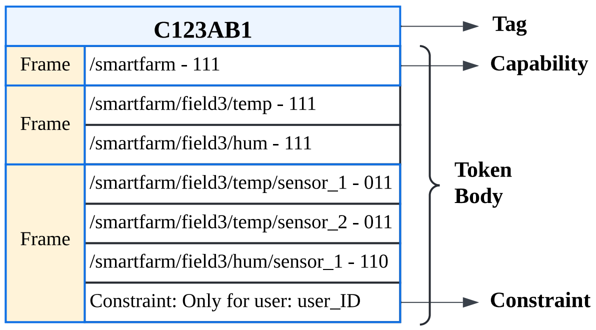

C123AB1
Frame
/smartfarm - 111
Frame
/smartfarm/field3/temp - 111
/smartfarm/field3/hum - 111
Frame
/smartfarm/field3/temp/sensor_1 - 011
/smartfarm/field3/temp/sensor_2 - 011
/smartfarm/field3/hum/sensor_1 - 110
Constraint: Only for user: user_ID
Tag
Capability
Token Body
Constraint

Fig. 1: FACTS Token

For fine-grained access control, we employ capability-based security [16]–[18]. Capability systems use unforgeable tokens attached to request messages to grant access to resources using the principle of least privilege (only the minimum level of access is granted). Tokens consist of a body and a tag. The body is composed of frames that contain information needed to implement a given security policy. To implement decentralized policies, a holder of a token can append one or more frames to that token to create a *derivation* which is verifiable and tamper-proof. When the token is presented to a service for verification of authorization, the service must verify that each frame in the token was correctly added and that each successive frame grants no additional privilege (i.e. each frame of a derivation *attenuates* privilege).

Each token is carries a tag which is the HMAC signature for the entire token. When a frame is added, the computation of the new tag depends on the previous tag for the token without the frame appended. A service issues a principal token to start the chain of frames and uses a private secret as the key from which the first tag is generated. Thus a service, when presented with a derivation of the principal token, and verify (by computing the intermediate tags from the principal token forward) that each frame is a valid attenuation.

It is possible to implement security policies that do not depend on a notion of identity using this mechanism. However, identity is intrinsic to IoT-cloud integration. To implement identity, a service that wishes to authenticate an identity must send a fully-attenuated derivation of a principal token to an identity service over a secure channel (e.g. using TLS). The identity service maintains a list of authorized users of this service. When a user authenticates to the identity service, it creates a derivation of the principal token that includes a verified identity ID and returns it to the service over a secure channel. To enable a device to access the service on behalf of the identity carried in the token, it installs the token in the device when the device is commissioned (e.g. via a secure channel or secure physical link).

The device can include the identity token as a frame in any request and the service can verify that the ID in the token was originally authenticated by the identity service. Since the identity token is not a strict attenuation of access rights, we encode them as *constraints* within a frame and check them as part of capability validation. Constraints contextually

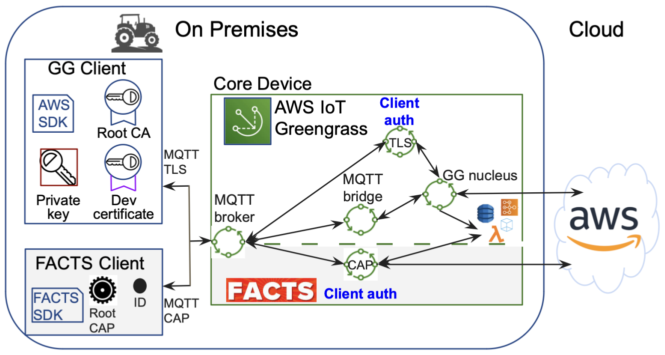


Fig. 2: Depiction of an example FACTS IoT deployment

attenuate tokens and thus enable arbitrary security policies to be specified. Note that each service implements its own interpretation of tokens presented to it. Thus it is possible for a service to accept requests from a device that includes the identity token from one user and other requests from the device on behalf of a different user.

Figure 1 depicts a FACTS token with three frames. Each frame names a resource identified by an MQTT topic (e.g. /smartfarm) and specifies a bit vector describing access rights `read`, `write`, and `execute` for the resource. The read bit indicates that the requester may subscribe to the topic. The write bit indicates that the data payload encoded in the message may be written to the resource. The execute bit indicates that the payload write can also trigger execution of a serverless function. The execute bit is not interpreted unless the write bit is set. Each frame attenuates the previous frame; the bottom-most frame in this example contains an identity constraint. The HMAC signature computed over the frame is included with the frame.

When a request comes in, FACTS extracts the capabilities from the token, validates the identity (if any), and authorizes the operation (by verifying the capabilities and constraints). Once authorized, FACTS logs the request (for auditing purposes) and executes the operation. FACTS uses the services deployed on the core devices to implement the operation, e.g. access to local storage or function invocation.

## *B. Deployment*

Figure 2 exemplifies a FACTS deployment. The gray sections of the figure represent FACTS services for interfacing devices to AWS Greengrass. Using FACTS, GG core devices use TLS to interact with the cloud directly and they use a customized combination of TLS and fine-grained access control for interoperation with other devices not hosting the GG core at the edge.

The figure depicts an IoT deployment consisting of two client devices (representing resource-constrained sensors or actuators), and a Greengrass (GG) core device, all co-located at the edge of the network (i.e. on-premises). We focus on secure edge device interoperation in this work. Client devices are IoT devices that connect to the GG core device using the MQTT protocol. Clients send data to the core for storage, processing, and function triggering via MQTT publish events; they receive commands from the core and data from other clients via MQTT subscription messages. Core resources map to MQTT topics.

A GG core device is an IoT device that proxies access to remote AWS cloud services and IoT device management for client devices; it also provides client devices with local functionality by executing software modules called *components* (depicted as circles and the right-most service icons in the core rectangle). The GG core itself is controlled using a mandatory component called a *nucleus* which runs continuously via a Java Virtual Machine (JVM) runtime system.

Component functionality that the core can implement includes MQTT message processing, data storage, data processing and machine learning services, as well as arbitrary functions (AWS supplied or user-defined) that are triggered by a local AWS Lambda component in response to MQTT publish events. When a GG client device connects to the GG core, the core attempts to authenticate and authorize the device (using the *TLS client auth* component in the figure) with the AWS cloud. If the GG core is not able to reach the cloud, it performs the authorization directly using locally cached credentials for the client. Note that at the time a client is first introduced, the GG core must have connectivity to the AWS cloud or its credentials cannot be cached.

The core performs mutual TLS authentication using X.509 certificates transmitted as part of an MQTT connection request from a client device. Clients can then publish MQTT messages over an encrypted channel until the connection is terminated. The core authorizes client operations triggered by MQTT to publish events using IAM [31] policies specified as part of the deployment configuration and encoded in the client certificate. The length of time that the core stores client credentials locally is a configurable deployment parameter. Client credentials required by MQTT TLS include a client (Dev) certificate, a private key, and the root certificate authority (RootCA; identity of client certificate signing authority). The MQTT client is responsible for data acquisition (for publishing), subscription management (for subscription) and setup/teardown of the MQTT TLS connection with the core.

FACTS extends both client and core device functionality (cf the gray sections of Figure 2). On the client, FACTS provides a simple software development kit (SDK) for Python. The script facilitates the same functionality as on the GG client (for publish and subscribe). It also sets up and tears down the MQTT connection (without TLS; MQTT CAP in the figure) and augments publish events with FACTS request capabilities. FACTS automatically generates request capabilities for each MQTT publish by combining and digitally signing the client's root capability (Root CAP), optionally a digital signature for the data being sent, and the client's identity token (ID). As with GG, these client credentials are passed to the device once via a secure channel from the identity provider during client/core commissioning. With FACTS, however, the identity provider can be co-located with the devices and core (precluding the need for an Internet connection to a centralized cloud). FACTS transmits the request capability with the data as part of an

MQTT publish message.

On the core, FACTS receives publish events via the MQTT broker. FACTS, implemented as a GG component (lower Client auth in the figure), verifies the request capability, and logs the requests (for auditing purposes). If the request is authorized, FACTS stores the data and optionally invokes other GG components on behalf of the client (providing the same functionality for client devices as GG alone).

## IV. Evaluation

We next evaluate FACTS by comparing its energy consumption to that of using Greengrass for IoT authentication and authorization for things (edge devices) using different scenarios. We first describe our experimental setup and then present our results. We refer to the combination of authentication and authorization as simply *Auth Services (AS)*.

### A. Experimental Setup

The core and client devices that comprise our IoT deployments are Raspberry Pi 3B+ single-board computers. These devices run the Raspberry Pi 32-bit OS (Bullseye) and have quad-core ARMv8 CPUs clocked at 1.4 GHz and 1 GB of RAM. The client devices connect to the core device via a private Ethernet connection. Each client sends data to the core via the MQTT (v5) protocol implemented using the Eclipse Paho MQTT Python client library v1.6.1.

The Greengrass core device (GG) runs AWS Greengrass version 2.12.1. AWS IoT requires that the GG core be connected to the Internet to interact with AWS IoT services during configuration and every 30 minutes of disconnection (in order to be able to continue to authenticate clients). We use a WiFi connection for the GG-to-cloud connectivity. During periods of disconnection, the GG core caches authentication-validating credentials so that for clients can interact with the core while the core is disconnected from the cloud. In the following experiments, once the GG core authenticates with the cloud, we disable the WiFi connection so that it does not consume energy or cause performance interference. During online configuration, we register each client device with AWS IoT and configure the necessary IAM policies to authorize access to the Greengrass core device. We install the X.509 certificates manually as described in the AWS IoT documentation.AWS IoT AS is performed by the core using a TLS layer and validation of the X.509 certificate chain. Once authenticated and authorized (via AWS security policies), client devices publish their messages using the encrypted channel created via this process. If a device disconnects, it must reconnect (and perform the TLS handshake again) to establish connectivity and to transmit additional data.

The FACTS core device implements its capability-based access control using CAPLets version 1.0 [16]. FACTS also integrates the mosquitto MQTT broker without TLS (note that the the Greengrass broker requires TLS) and instead controls access to resources using capabilities. An MQTT subscriber, written in C, receives published messages from devices, extracts the capability from each, then validates the request. If successful (i.e. the request is authenticated and authorized), the service posts the data to a log and optionally invokes a serverless/FaaS function. We install the root capability (principal capability) and identity token on each client device manually for use with the FACTS core as part of device commissioning (as described previously). FACTS clients use MQTT (v5) over an *unencrypted channel* to send their data and a request token with every publish event. FACTS performs AS on every request (MQTT publish).

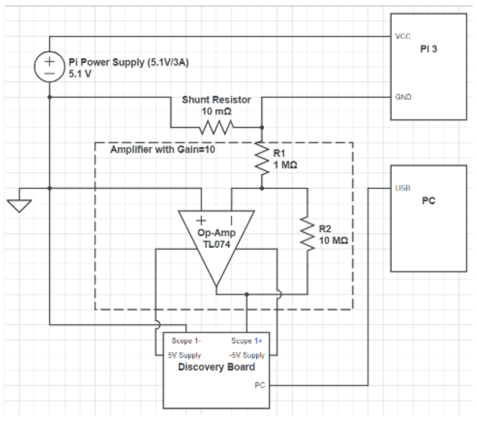


Fig. 3: Experimental setup for measuring power consumption

Note that the channel is secure but not private. That is, an eavesdropper on the network can capture the message traffic. However, because FACTS signs the data, an attacker cannot tamper with it and replay the message with a different data payload. The full CAPLets protocol includes replay protection (i.e. encrypted sequence numbers) for individual messages and support for perfect-forward secrecy, but these features are disabled in the implementation we tested in these experiments.

It is surprisingly difficult to measure accurately the energy consumption of Pi devices [32], [33]. To do so, we use an Analog Discovery Board 2 [34] and have implemented a digital circuit to make this possible. Our circuit is equipped with a shunt resistor in series with the Pi which measures current. Figure 3 presents the circuit diagram we use. We collect the data using a sample rate of 100 Hz using a laptop which we post-process to generate our results. We calibrate the measurements using a USB energy monitor (as ground truth) between the board and the circuit prior to running the experiments. We use this device to measure the voltage and current during the experiment. We then compute average power (in watts), time duration (in seconds), and energy consumption (in joules) from this data for the time duration of each experiment. All devices in the deployment are time synchronized using NTP.We perform each experiment 5 times and compute the average across runs. We collect performance data for both a client device and a core device which we present in the following subsection.

### B. Results

We present performance comparisons between FACTS and Greengrass from the perspective of both a client device and a core device. In our IoT deployments, both core and client

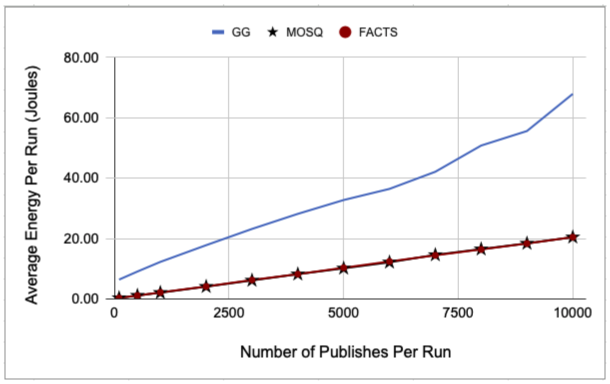


Fig. 4: Average energy consumption (lower is better) for different publish counts per experiment using the GG MQTT broker, the mosquitto MQTT broker (MOSQ) and FACTS (which also uses mosquitto).

devices are battery powered. Client devices are sensors which sample data from their environment and send it to the core via MQTT. The core then stores the data (which optionally triggers a function) for later analysis.

The results presented herein are generated using MQTT QoS 1. We use QoS 1 because we found that the Greengrass MQTT broker dropped many publish messages if more than 1000 were sent at a time (FACTS surprisingly dropped none even when over 10000 messages sent at once). We also measured the performance difference for both core implementations (FACTS and Greengrass) for 100 messages using both QoS levels and found the results to be similar (for duration, average power, and energy consumption). Thus we omit data for QoS 0 for brevity and consider only QoS 1 in our evaluation.

## C. Client Performance

We first evaluate the performance impact of using GG and FACTS on the client side. Figure 4 graphs the energy consumption of the client for a set of experiments. For each experiment, the client publishes a specified number of messages to the MQTT broker at the core. We consider counts of 50 to 10000 publish events. The client connects to the broker once at the start of each event (and disconnects when it receives an acknowledgment from the last publish).

The blue, unmarked curve shows the average energy consumption in joules for a client communicating with a GG core device (which implements its own MQTT broker in Java). FACTS uses mosquitto as its MQTT broker because it is implemented in C/C++ and allows clients to communicate with the broker with or without TLS (the GG broker requires that TLS be used). Energy is depicted on the $y$-axis and back-to-back message count (for 8-byte data payloads in each message) is shown on the $x$-axis. For each set of messages, the client makes a single connection with the broker. FACTS energy consumption is shown via the red curve with circle markers. On average, the client consumes 74% less energy interacting with FACTS than when interacting with GG.

| Pub Count | Duration (s) | Avg Power (W) | Energy (J) |
|---|---|---|---|
| 100 | 3.83 | 1.67 | 6.42 |
| 500 | 5.34 | 1.70 | 9.08 |
| 1000 | 7.21 | 1.70 | 12.27 |
| 2000 | 10.24 | 1.74 | 17.81 |
| 3000 | 13.25 | 1.75 | 23.19 |
| 4000 | 15.97 | 1.76 | 28.18 |
| 5000 | 18.50 | 1.77 | 32.76 |
| 6000 | 20.55 | 1.77 | 36.45 |
| 7000 | 23.45 | 1.80 | 42.13 |
| 8000 | 28.14 | 1.80 | 50.80 |
| 9000 | 31.00 | 1.79 | 55.61 |
| 10000 | 38.30 | 1.78 | 67.91 |

Fig. 5: Average publish performance for client-side GG

| Pub Count | Duration (s) | Avg Power (W) | Energy (J) |
|---|---|---|---|
| 100 | 0.16 | 1.83 | 0.34 |
| 500 | 0.55 | 1.97 | 1.12 |
| 1000 | 1.05 | 2.01 | 2.16 |
| 2000 | 2.03 | 2.01 | 4.14 |
| 3000 | 3.06 | 2.03 | 6.27 |
| 4000 | 4.06 | 2.01 | 8.21 |
| 5000 | 5.08 | 2.02 | 10.33 |
| 6000 | 6.09 | 2.03 | 12.40 |
| 7000 | 7.08 | 2.04 | 14.46 |
| 8000 | 8.06 | 2.03 | 16.41 |
| 9000 | 9.03 | 2.03 | 18.36 |
| 10000 | 10.05 | 2.03 | 20.46 |

Fig. 6: Average publish performance for client-side FACTS

To determine if FACTS introduces any overhead on the client side, we also measure the use of the mosquitto MQTT broker without FACTS. This data is shown using a black curve with star markers (MOSQ). MOSQ and FACTS align and are virtually indistinguishable in the figure indicating that FACTS introduces almost no overhead on the client side. Quantitatively, we find that there is an average difference of $0.03$ joules between MOSQ and FACTS. This low overhead is likely to be similar for any broker that FACTS integrates (e.g. if it were possible to use the GG broker directly but without TLS). The difference in broker performance (without FACTS) is likely due to the use of the Java programming language for the GG broker (vs C/C++ for mosquitto) or the use of encryption. It is not due to the TLS connection overhead since the MQTT connection latency is amortized over all messages in each individual experiment.

We present the raw measurement data in Figures 5 (for GG) and 6 (for FACTS). The columns show the publish count, experiment duration in seconds, average power in watts, and average energy consumption (across runs) in joules. Overall, the client when using FACTS is significantly faster than when using GG and the average power is slightly higher. Due to the reduction in time, energy consumption using FACTS is far less even for very large numbers of publish events.

These improvements occur even though FACTS sends a larger payload (530 more bytes per publish on average) per message to encode the capability in the request. However, it also avoids encrypting every message as GG does. By trading off privacy (data is sent between the client devices and the core in the clear), FACTS achieves much lower energy consumption at the client side than GG.

## D. Core Performance

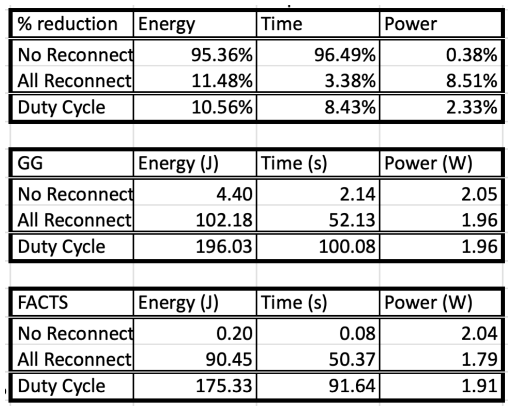
The difference between using FACTS and GG on the client side is primarily due to the broker implementation. Thus, we next consider the energy consumption for GG and FACTS for the core device. In this study, we use the same core device for both GG and FACTS. For each set of experiments however, we turn off any services of the competitor (in addition to the WiFi as discussed above) to avoid unnecessary energy use or performance interference.

We first consider the cost of client connections when using GG and FACTS for different experimental scenarios. In this evaluation, we consider three scenarios. In the first scenario (`No Reconnect`), AS is performed once by GG when the connection is initiated by a client device. This scenario represents the case in which the cost of AS is amortized over many publish events. Note that for FACTS, every publish message includes the request capability which FACTS validates prior to data storage or function invocation. That is, FACTS implements *per-message authentication and authorization* as opposed to GG which relies on a TLS session to communicate otherwise unauthenticated messages. As a result FACTS does not amortize authentication/authorization over multiple messages but each capability check is significantly less resource intensive compared to using TLS. Moreover, the data in the FACTS publish event is sent is clear text (unencrypted) trading off data privacy for energy efficiency when privacy is not required. It is tamper-proof, authorized, and all operations are authenticated, but it is not channel private.

In the second scenario (`All Reconnect`), clients connect and disconnect on every publish. This represents the scenario in which clients disconnect frequently to allow event-driven "deep sleep" as a way of conserving battery power or to "batch" the data to reduce message count. For such a setting however, GG introduces overhead for both connection establishment and TLS for every message. FACTS imposes overhead only for connection establishment. For both of these scenarios, the client publishes 50 messages. Thus, there are 50 connects for `All Reconnect` and 1 connect for `No Reconnect`.

The third scenario (`Duty Cycle`) represents a more realistic use case which we include for comparison. In this scenario, the client devices connect and send data for a period of time and then sleep waiting until the next set of measurements becomes available or to conserve battery life. In our experiments, the client sends 100 messages then sleeps for 10 seconds. It repeats this 10 times for a total of 1000 publish events across 10 connections.

Figure 7 presents three tables of data for these scenarios and experiments. Each table has columns for average energy consumption in joules per experiment (Energy), average experiment duration in seconds (Time), and average power in watts (Power). Each row shows a set of results for the three scenarios: `No Reconnect`, `All Reconnect`, and `Duty Cycle`. The topmost table shows the performance improvement as a percentage when we use FACTS vs GG.

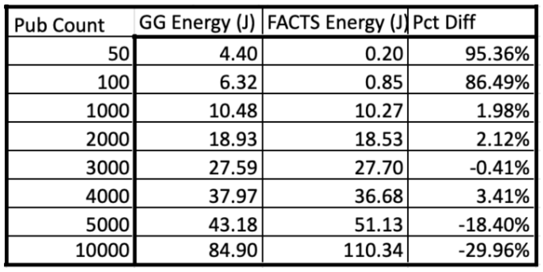


| % reduction | Energy | Time | Power |
|---|---|---|---|
| No Reconnect | 95.36% | 96.49% | 0.38% |
| All Reconnect | 11.48% | 3.38% | 8.51% |
| Duty Cycle | 10.56% | 8.43% | 2.33% |

| GG | Energy (J) | Time (s) | Power (W) |
|---|---|---|---|
| No Reconnect | 4.40 | 2.14 | 2.05 |
| All Reconnect | 102.18 | 52.13 | 1.96 |
| Duty Cycle | 196.03 | 100.08 | 1.96 |

| FACTS | Energy (J) | Time (s) | Power (W) |
|---|---|---|---|
| No Reconnect | 0.20 | 0.08 | 2.04 |
| All Reconnect | 90.45 | 50.37 | 1.79 |
| Duty Cycle | 175.33 | 91.64 | 1.91 |

Fig. 7: Connection Cost at the core device. The top table shows the percent reduction in energy, time, and average power enabled by FACTS vs GG. The middle and lower tables show the raw data for each core implementation.



| Pub Count | GG Energy (J) | FACTS Energy (J) | Pct Diff |
|---|---|---|---|
| 50 | 4.40 | 0.20 | 95.36% |
| 100 | 6.32 | 0.85 | 86.49% |
| 1000 | 10.48 | 10.27 | 1.98% |
| 2000 | 18.93 | 18.53 | 2.12% |
| 3000 | 27.59 | 27.70 | -0.41% |
| 4000 | 37.97 | 36.68 | 3.41% |
| 5000 | 43.18 | 51.13 | -18.40% |
| 10000 | 84.90 | 110.34 | -29.96% |

Fig. 8: Energy in joules for publish events at core device.

The middle table shows the raw data for GG and the lower table shows the raw data for FACTS .

This data shows that the cost of establishing a connection is high for both GG and FACTS. On average for the `All Reconnect` scenario, a connection costs 2.04J and 1.04s for GG and 1.81J and 1.01s for FACTS. This results in a 95% reduction in energy consumption for FACTS vs GG. The bulk of this improvement comes from reducing the time associated with each connection. In comparison, the average for the `No Reconnect` scenario, a connection costs 0.088J and 0.043s for GG and 0.004J and 0.002s for FACTS. This results in a 11% reduction in energy consumption for FACTS vs GG.

For the more realistic use case (`Duty Cycle`), FACTS uses 11% less energy, 8% less time, and 2% less power than GG. Note that because the core never sleeps (it must be available for connections from clients), it is unable to reduce this overhead when clients are not communicating with it. We are considering new ways of reducing the power consumption of core devices at the edge as part of future work.

Figure 8 presents table data for the `No Reconnect` scenario. It enables us to evaluate the overhead of publish events at the core without the overhead of connection (a single connect occurs as the start of the publish process). The first column shows the number of publish messages sent by the client for each experiment. The second column shows the energy consumption in joules for GG on average across experiments. The third column shows the energy consumption in joules for FACTS on average across experiments. The last column shows the percent reduction (or increase) in energy consumption over GG that FACTS enables.

For small numbers of publish events, FACTS significantly outperforms GG. For publish counts between 1000 and 4000, FACTS performs similarly to GG at the core device. When the number of publishes at once becomes very large (greater than 5000 per experiment), we observe a performance cross over between GG and FACTS. GG outperforms FACTS by amortizing the cost of encryption and TLS communication establishment across the many messages. FACTS must validate the capability for every message and adds communication overhead to each message (for the larger payload). This processing time starts to accumulate when there are more than 5000 messages sent at once.

This result speaks to the trade offs associated with each type of AS mechanism. As part of future work, we are considering how to combine the use of FACTS and GG adaptively over time depending on data size and messaging and security requirements. Our goal is to use the AS mechanism that will achieve the lowest energy use dynamically depending on the application requirements.

## V. Conclusions and Future Work

We present FACTS – hybrid access control for fog/edge devices that integrates low overhead capability-based security with AWS Greengrass. By combining TLS with the use of capabilities, we enable IoT deployment administrators to decide where and when to use encrypted channels to achieve the best trade-off between data privacy performance, and energy efficiency. Our evaluation shows that FACTS is able to achieve end-to-end message based authorization with significantly lower overhead than using AWS Greengrass alone.

As part of future work, we are investigating more energy efficient implementations of the MQTT broker and reducing the size of the FACTS's capability. We are also developing the necessary tooling to automate IoT deployment that developers can use to specify which devices use TLS and which use capability-based security for an IoT application.

## References

[1] Microsoft, "Azure IoT Edge," 2018, "https://azure.microsoft.com/en-us/services/iot-edge/" Acc. May 2018.

[2] "Alibaba IoT Edge," 2024, "https://www.alibabacloud.com/product/link-iotedge" Acc. 29-Mar-2024.

[3] 2018, "https://docs.aws.amazon.com/iot".

[4] "AWS Greengrass," "https://aws.amazon.com/greengrass/".

[5] "MQTT," https://mqtt.org/, 2018.

[6] "AWS Lambda," https://aws.amazon.com/lambda/, 2017.

[7] ZigBee Alliance, "ZigBee Alliance," https://www.zigbee.org/.

[8] I. Froiz-Míguez, T. M. Fernández-Caramés, P. Fraga-Lamas, and L. Castedo, "Design, implementation and practical evaluation of an iot home automation system for fog computing applications based on mqtt and zigbee-wifi sensor nodes," *Sensors*, vol. 18, no. 8, p. 2660, 2018.

[9] B. Fouladi and S. Ghanoun, "Security evaluation of the z-wave wireless protocol," *Blackhat USA*, vol. 24, pp. 1–2, 2013.

[10] S. Žitnik, M. Janković, K. Petrovčič, and M. Bajec, "Architecture of standard-based, interoperable and extensible iot platform," in *2016 24th Telecommunications Forum (TELFOR)*. IEEE, 2016, pp. 1–4.

[11] J. de Carvalho Silva, J. J. Rodrigues, A. M. Alberti, P. Solic, and A. L. Aquino, "Lorawan—a low power wan protocol for internet of things: A review and opportunities," in *International multidisciplinary conference on computer and energy science (SpliTech)*, 2017.

[12] A. N. Rosli, R. Mohamad, Y. W. M. Yusof, S. Shahbudin, and F. Y. A. Rahman, "Implementation of mqtt and lorawan system for real-time environmental monitoring application," in *Symposium on Computer Applications & Industrial Electronics*, 2020.

[13] S. Turner, "Transport layer security," *IEEE Internet Computing*, vol. 18, no. 6, 2014.

[14] M. Suárez-Albela, T. M. Fernández-Caramés, P. Fraga-Lamas, and L. Castedo, "A practical performance comparison of ecc and rsa for resource-constrained iot devices," in *Global IoT Summit*, 2018.

[15] Q. Fan, J. Chen, M. Shojafar, S. Kumari, and D. He, "Sake*: A symmetric authenticated key exchange protocol with perfect forward secrecy for industrial internet of things," *IEEE transactions on industrial informatics*, vol. 18, no. 9, pp. 6424–6434, 2022.

[16] F. Bakir, C. Krintz, and R. Wolski, "CAPLets: Resource Aware, Capability-Based Access Control for IoT," in *ACM Symposium on Edge Computing*, 2021.

[17] A. Birgisson, J. G. Politz, U. Erlingsson, A. Taly, M. Vrable, and M. Lentczner, "Macaroons: Cookies with contextual caveats for decentralized authorization in the cloud," in *Network and Distributed System Security Symposium*, 2014.

[18] S. Mullender, G. van Rossum, A. Tanenbaum, R. van Renesse, and H. van Staveren, "Amoeba – A distributed Operating System for the 1990's," *IEEE Computer*, vol. 23, no. 5, May 1990.

[19] J. Hughes and W. Diffie, "The challenges of iot, tls, and random number generators in the real world," *ACM Queue*, vol. 20, 2022.

[20] M. T. Paracha, D. J. Dubois, N. Vallina-Rodriguez, and D. Choffnes, "Iotls: understanding tls usage in consumer iot devices," in *Proceedings of the 21st ACM Internet Measurement Conference*, 2021, pp. 165–178.

[21] R. Trimananda, A. Younis, B. Wang, B. Xu, B. Demsky, and G. Xu, "Vigilia: Securing smart home edge computing," in *Symposium on Edge Computing (SEC)*, 2018.

[22] X. W. Pengkun Li, Jinshu Su, "itls: Lightweight transport-layer security protocol for iot with minimal latency and perfect forward secrecy," *IEEE Internet of Things Journal*, 2020.

[23] S. Pal, M. Hitchens, V. Varadharajan, and T. Rabejaja, "On design of a fine-grained access control architecture for securing iot-enabled smart healthcare systems," in *EAI International Conference on Mobile and Ubiquitous Systems: Computing, Networking and Services*, 2017.

[24] S. Pal and Z. Jadidi, "Protocol-based and hybrid access control for the iot," *Sensors*, vol. 21, no. 6832, 2021.

[25] H. Pranata, R. Athauda, and G. Skinner, "Securing and governing access in ad-hoc networks of internet of things," in *International Conference on Engineering and Applied Science*, 2012.

[26] S. Pal, M. Hitchens, V. Varadharajan, and T. Rabehaja, "Policy-based access control for constrained healthcare resources in the context of the internet of things," *J. Netw. Comput. Appl*, vol. 139, 2019.

[27] P. Colombo and E. Ferrari, "Access control enforcement within mqtt-based internet of things ecosystems," in *Symposium on Access Control Models and Technologies*, 2018.

[28] A. Hintaw, S. Manickam, S. Karuppayah, and M. F. Aboalmaaly, "A brief review on mqtt's security issues within the internet of things (iot)," *Communications*, vol. 14, no. 6, 2019.

[29] G. Selander, J. Mattsson, and F. Palombini, "Ephemeral diffie-hellman over cose (edhoc)," IETF, Tech. Rep. 9528, Mar 2024.

[30] G. Selander, J. Mattsson, F. Palombini, and L. Seitz, "Object security for constrained restful environments (oscore)," IETF, Tech. Rep. 8613, Jul 2019.

[31] "AWS IAM," http://aws.amazon.com/iam/; accessed May 31, 2024.

[32] M. D. Mudaliar and N. Sivakumar, "Iot based real time energy monitoring system using raspberry pi," *Internet of Things*, vol. 12, 2020.

[33] G. Bekaroo and A. Santokhee, "Power consumption of the raspberry pi: A comparative analysis," in *IEEE International Conference on Emerging Technologies and Innovative Business Practices for the Transformation of Societies*, 2016.

[34] M. Dabacan, "Analog discovery 2 reference manual," *Analog Discovery 2 Reference Manual-Digilent Reference*, 2018.