An Internet of Things Software and Firmware Update Architecture Based on the SUIT Specification

Simon Carlson

Agenda

- Introduction
- Background
- Method
- Results
- Conclusions
- Questions

Introduction

Massive growth in IoT expected, cyberattacks increasing both in amount and severity

SUIT - Software Updates for Internet of Things

“How can the SUIT specification be applied to develop a technology agnostic and interoperable update architecture for heterogeneous networks of IoT devices?”

Purpose: Aid other projects securing their products

Agenda

- ~~Introduction~~
- Background
- Method
- Results
- Conclusions
- Questions

Background - SUIT

Aims to define a firmware update solution for IoT devices that is interoperable and non-proprietary

Does not try to define new transport, discovery, or security mechanisms

Covers both architecture for transporting updates and needed metadata

Background - SUIT architecture

Agnostic to how firmware images are distributed

Friendly to broadcast delivery

Use state-of-the-art security mechanisms

Rollback attacks must be prevented

High reliability

Operate with a small bootloader

Small parser

Minimal impact on existing firmware formats

Robust permission

Operating modes

Background - SUIT information model

Presents the elements of a manifest that is needed to securely perform an update

Trade off between flexibility and size: not all elements are always needed but completely removing them eliminates certain use cases

Example elements: version number, sequence number, payload format, size

Agenda

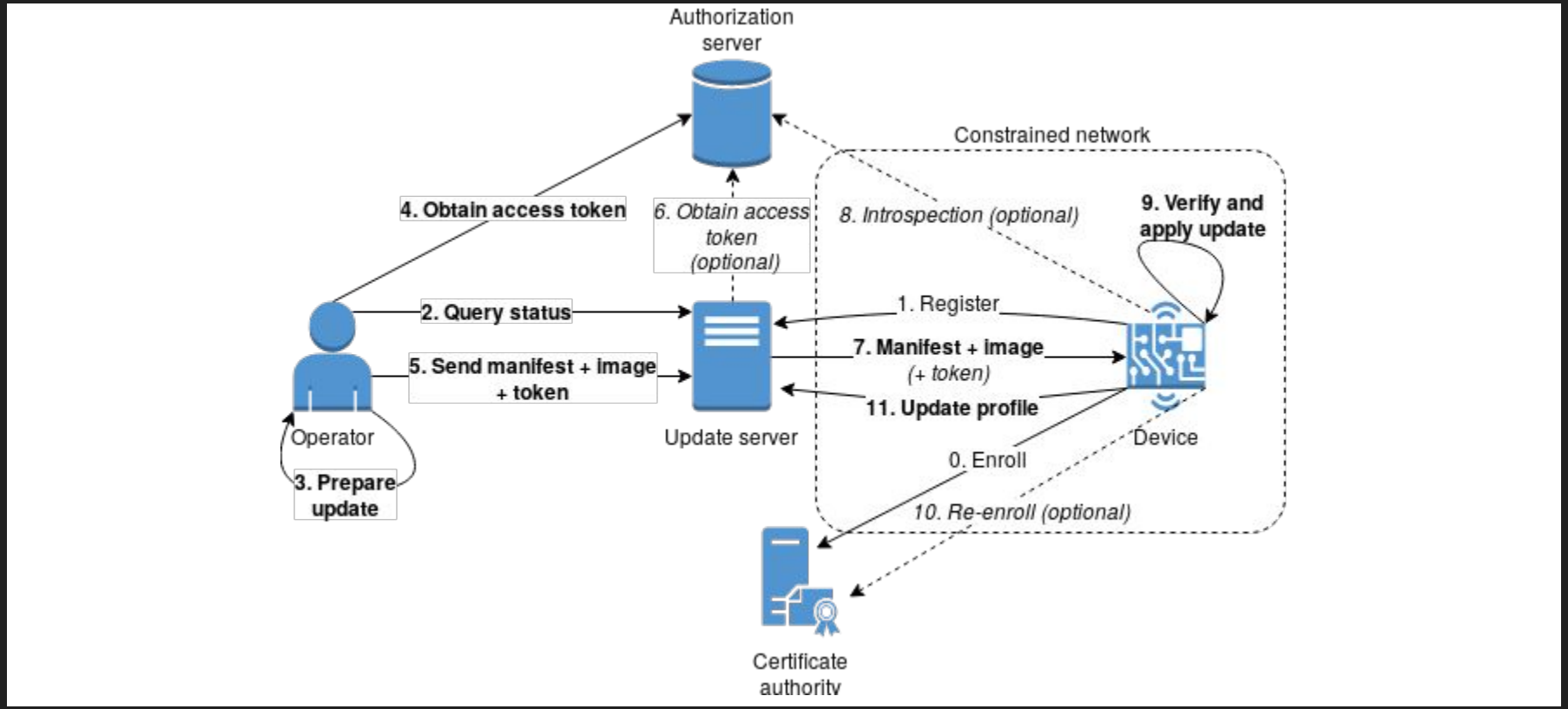
- ~~Introduction~~
- ~~Background~~
- Method
- Results
- Conclusions
- Questions

Method

The proposed architecture defines five key areas:

- Roles of devices, servers, and operators
- Key management
- Device profiles and communication
- Authorization
- Update handling

Method - proposed architecture



Method - roles

Devices: run applications, can ask for and receive updates

Update servers: repositories for updates and device profiles, communicates with devices

Operators: checks devices status at server, creates and signs updates

Operators, update servers, and devices are all enrolled

Method - key management

Symmetric keys do not scale well

An enrolling device needs to trust the certificate authority

After updating, make sure certificates still are valid

Keys for signing payloads

Method - device profiles

Different IoT devices might mean different protocols, how to reach devices?

Devices register at update servers after enrollment and update

Also serves the operator when creating updates and choosing targets

Method - authorization

Devices keep whitelists of update servers and operators. Coarse-grain access control

Access tokens to define a scope of a particular resource. Enables many use cases

Operators always use tokens with servers as they are both typically unconstrained

Servers may use tokens with devices, but might not always be applicable

Method - update handling

Image must be decrypted and have its digest checked

Manifest must be stored for verifying image on boot and comparing sequence numbers

Options when storing the image: encrypted or unencrypted. Same for manifest

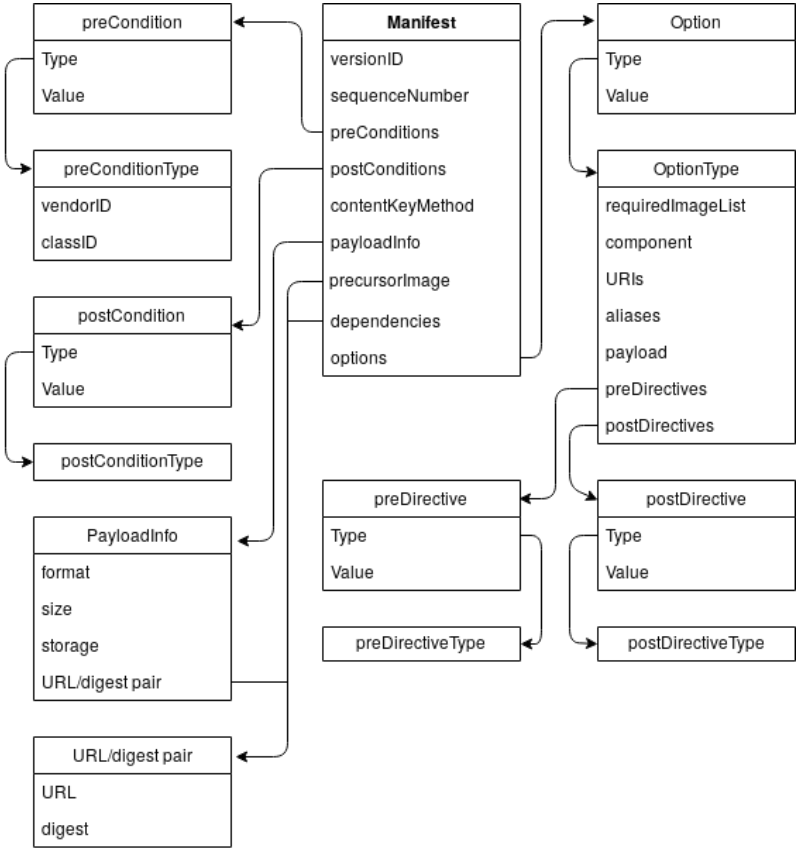
Method - manifest format

SUIT mentions manifest elements but no format

The manifest format should be:

- compliant with the SUIT information model
- as small as possible
- easy to parse
- extensible

Method - manifest format



Method - prototype implementation

Uses a DTLS/CoAP profile

Implemented both server and client in Contiki-NG, runs on Firefly boards

Pull model

Hardcoded manifest, deterministically generated data instead of actual image

Limitations: pre-shared keys instead of certificates, no tokens, payload encryption instead of signing

Agenda

- ~~Introduction~~
- ~~Background~~
- ~~Method~~
- Results
- Conclusions
- Questions

Results - architecture

Agnostic to how firmware images are distributed

Friendly to broadcast delivery

Use state-of-the-art security mechanisms

Rollback attacks must be prevented

High reliability

Operate with a small bootloader

Small parser

Minimal impact on existing firmware formats

Robust permission

Operating modes

Results - information model

Monotonic sequence numbers

Vendor and device-type identifiers

Best-before timestamps

Cryptographic authenticity

Authenticated payload type

Authenticated payload storage location

Authenticated remote resource location

Authenticated precursor images

Authenticated vendor and class IDs

Secure boot

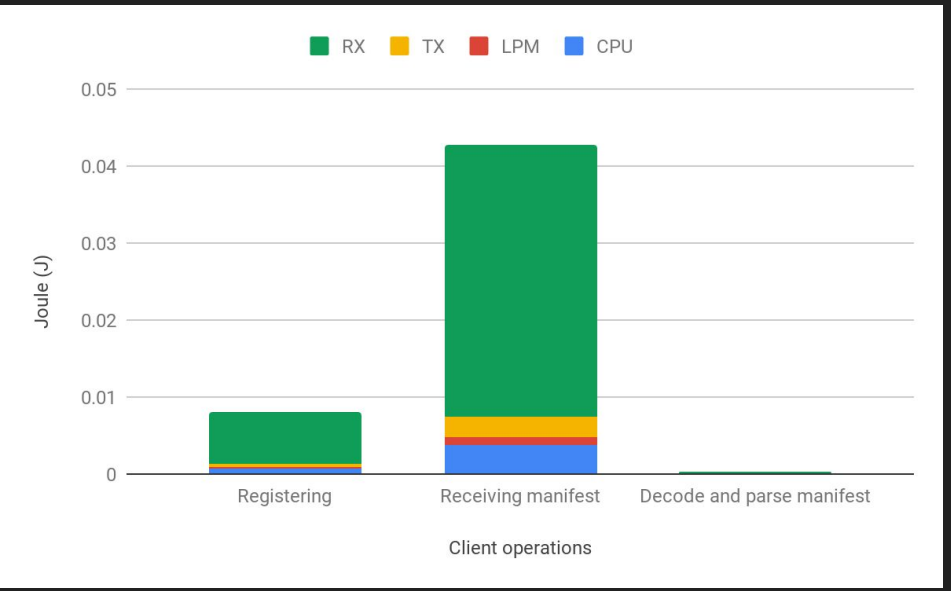
Rights require authenticity

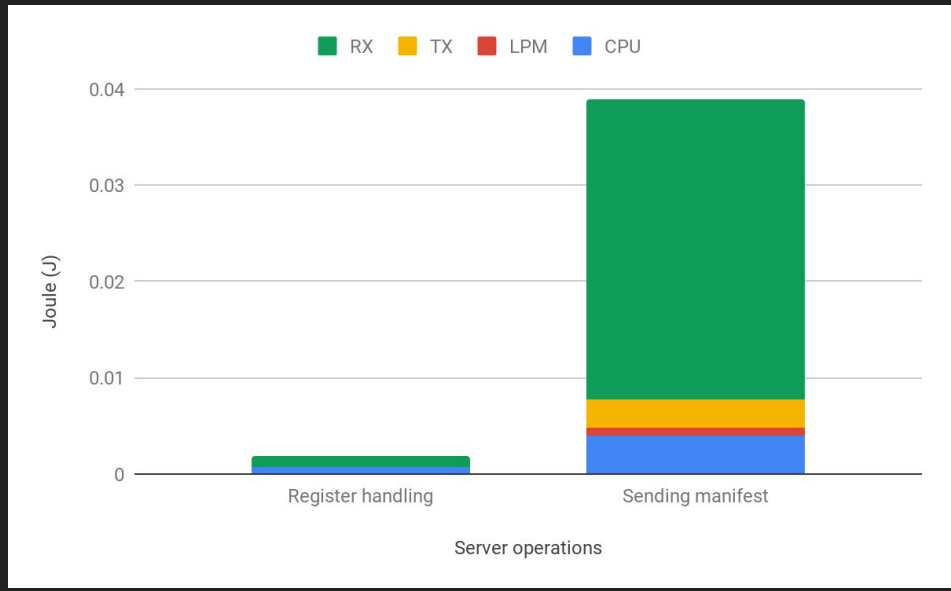
Firmware encryption

Encrypted manifests

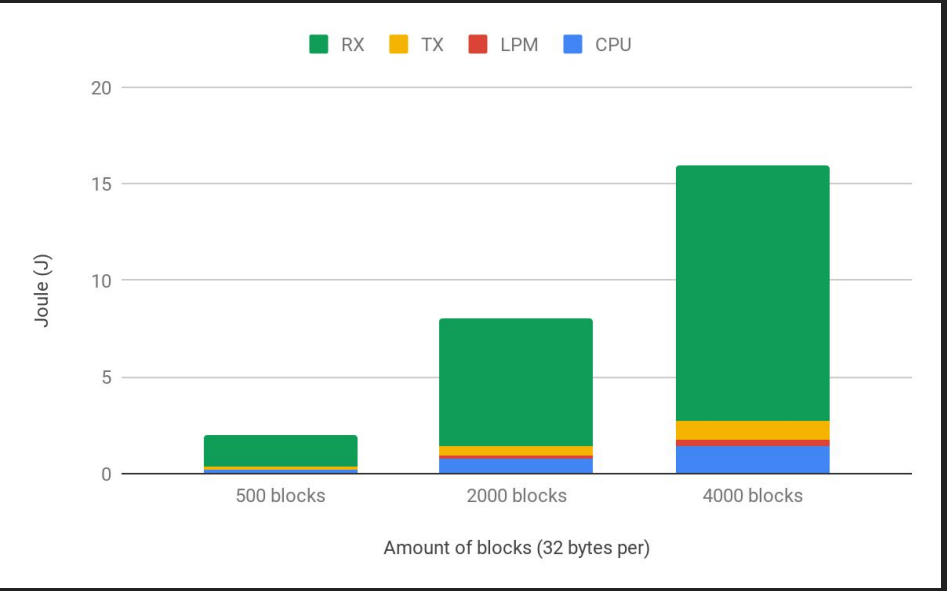
Access control list

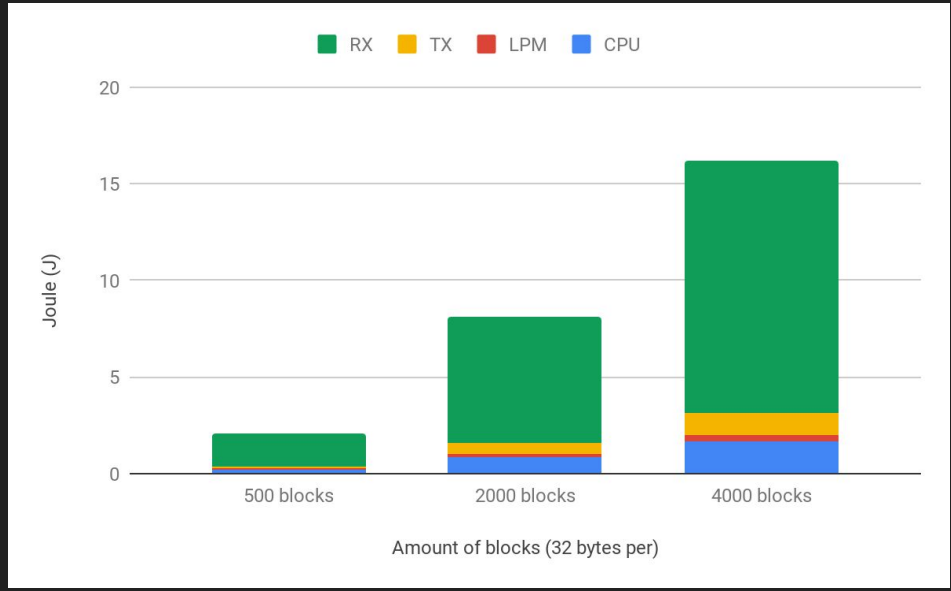
Results - energy consumption



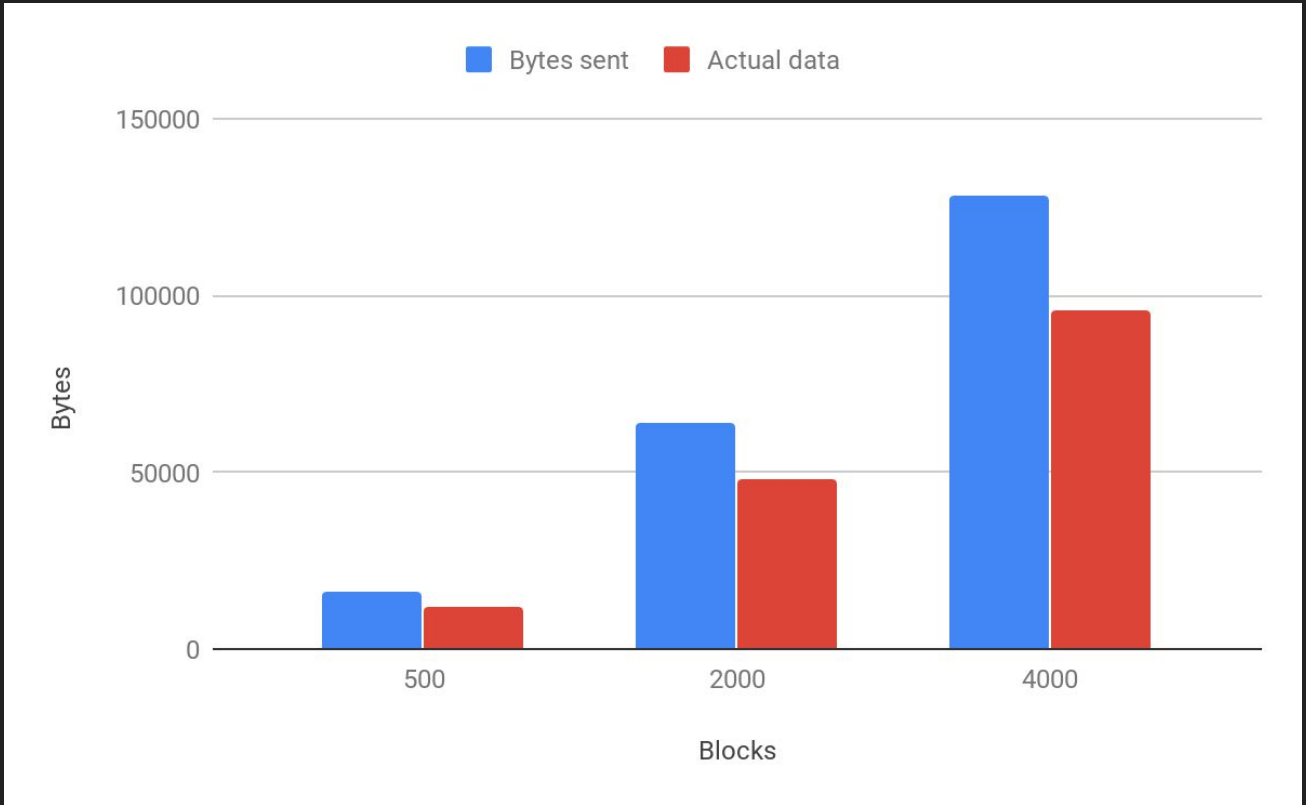


Results - energy consumption





Results - communication overhead



Results - code size

text	data	bss	dec	hex	filename
69982	1772	11303	85057	14472	bare-bones.elf
75956	1788	13599	91343	164cf	update-client.elf
77924	1884	11835	91643	165fb	update-server.elf

```
#include "contiki.h"
#include "contiki-net.h"
#include "coap-engine.h"
#include "coap-blocking-api.h"
#include "coap-keystore-simple.h"
PROCESS(bare_bones_process, "Bare bones process");
AUTOSTART_PROCESSES(&bare_bones_process);
PROCESS_THREAD(bare_bones_process, ev, data) {
        PROCESS_BEGIN();
        PROCESS_END();
}
```

Agenda

- ~~Introduction~~
- ~~Background~~
- ~~Method~~
- ~~Results~~
- Conclusions
- Questions

Discussion - conclusions

The architecture fulfills the requirements formulated by SUIT (with the exception of bootloader specific requirements), thus is a way of applying it

Fundamental requirements are part of the core architecture while some are implementation specific but prepared for

Flexible architecture aiming for robust implementations that make sense in specific deployments, reducing sizes of manifests

Prototype implementation to contextualize the architecture, seems feasible using Contiki-NG and Firefly devices

Discussion - security considerations

Timing of updates - hard to properly time an update, critical operations might be interrupted

Redundant devices - no support for updating redundant devices performing the same task

Physical threats - architecture (and SUIT) does not account for any physical threat

Importance of CA - single point of failure for communication

Battery draining attacks - push model always initiates manifest parsing at client

Agenda

- ~~Introduction~~
- ~~Background~~
- ~~Method~~
- ~~Results~~
- ~~Conclusions~~
- Questions

Questions?