UNIVERSITATEA POLITEHNICA DIN BUCUREȘTI
FACULTATEA DE AUTOMATICĂ ȘI CALCULATOARE
DEPARTAMENTUL DE CALCULATOARE

Sumar Teză Doctorat

Securizarea Infrastructurilor IoT Eterogene

Ing. Ioana-Maria Culic

Coordonator științific:

Prof. Dr. Ing. Răzvan-Victor Rughiniș

BUCUREȘTI
2023

CUPRINS

1 Introducere **1**

2 Enforsarea securității prin limbaje de programare moderne **5**

2.1 Integrarea limbajului de programare D în dispozitivele IoT cu restricții 5

2.2 Dezvoltarea de aplicații IoT sigure folosind JavaScript 6

2.2.1 Executarea JerryScript pe NXP Rapid IoT Prototyping Kit 7

2.2.2 Compilarea și integrarea JerryScript 7

2.2.3 Teste și rezultate 8

2.3 Programare bazată pe fluxuri pentru infrastructuri IoT 9

2.3.1 Implementarea platformei 9

2.3.2 Implementarea aplicației și rezultatele obținute 10

3 Dezvoltarea securizată a sistemelor IoT **11**

3.1 Instrumente de dezvoltare și prototipare IoT 11

3.1.1 Prezentare generală a platformei 11

3.1.2 Întreținerea platformei 14

3.1.3 Utilizarea platformei 16

3.2 Educația în domeniul IoT 17

3.2.1 Platformă hardware pentru abstractizarea circuitelor 17

3.2.2 Platformă de simulare hardware 20

4 Infrastructură de actualizare securizată pentru sistemele IoT **22**

4.1 Constrângerile platformei 22

4.2 Modelul matematic propus 23

4.2.1 Terminologie 23

4.2.2 Mulțimi 24

4.2.3 Modelul matematic 24
4.3 Implementarea și validarea modelului - IoTWay 25
4.4 Studiu de caz 27
4.5 Rezultate 27
4.5.1 Caracteristicile software-ului 27
4.5.2 Performanța instalărilor 27
4.5.3 Comparare performanțelor cu Balena 28

5 Securitatea la nivelul kernelui **30**
5.1 Aplicarea securității în Tock 30
5.2 Abordarea Propusă 31
5.3 Integrarea Sandbox-ului eBPF în Tock 31
5.3.1 Executorul eBPF 32
5.3.2 Preprocesarea bytecode-ului 33
5.3.3 Agentul de comunicare din spațiul utilizator cu executorul eBPF . . . 34
5.4 Teste și rezultate 34
5.4.1 Evaluarea implementării originale 35
5.4.2 Evaluarea abordării propuse 37
5.4.3 Rezultate 39

6 Concluzii **41**

Bibliografie **43**

ABSTRACT

Pe măsură ce tehnologiile Internet of Things (IoT) au devenit tot mai populare, numărul atacurilor asupra dispozitivelor IoT a crescut proporțional cu integrarea acestor tehnologii în diverse domenii. Astfel, securitatea a devenit un aspect esențial în dezvoltarea produselor IoT, mai ales de la apariția unor atacuri populare, precum Mirai Botnet. Infrastructurile IoT sunt sisteme complexe și eterogene, bazate pe o stivă hardware și software. Rezultatul este că mecanismele de securitate pentru sistemele Internet of Things trebuie să urmeze această stivă și să adreseze toate nivelele pentru a putea obține un produs cu un nivel mărit de securitate.

Scopul acestei teze este să analizeze mecanismele și politicile de securitate care pot fi implementate la diverse nivele din stivă și să propună soluții pentru a crește gradul de securitate a produselor finale. Contribuțiile prezentate adresează sisteme software de securitate, bazate pe mecanisme hardware pre-existente.

Pentru început, abordăm securitatea aplicațiilor Internet of Things, cu accent pe limbajele de programare pe care dezvoltatorii le pot folosi. Astfel, am identificat JerryScript, un motor de rulare a limbajului JavaScript pentru dispozitive integrate, ca cea mai potrivită abordare în dezvoltarea de aplicații sigure.

Mai departe, ne concentrăm pe instrumentele necesare pentru a facilita un proces de dezoltare a aplicațiilor IoT sigure. În această teză, propunem o implementare a unei platforme de dezvoltare complexă, care abstractizează interacțiunea cu dispozitivele hardware și care le permite dezvoltatorilor să se concentreze pe scrierea unor aplicații sigure. De semenea, am analizat provocările specifice procesului de predare a tehnologiilor IoT din prisma uneltelor de dezvoltare. Astfel, am propus un o platformă software pe care profesorii o pot folosi pentru a integra mai ușor sistemele IoT în curriculă. Rezultatul este o o platformă folosită de peste 30.000 de pasionați de IoT și profesori.

La un nivel mai înalt al stivei, am evidențiat o serie de lipsuri de securitate privind instalarea și aducerea la zi a aplicațiilor pentru sistemele IoT. Mai specific, am identificat lipsa unei infrastructuri deschise care să suporte aceste operațiuni într-un mod sigur. În acest context, contribuția noastră constă în propunerea unui model matematic pentru o astfel de infrastructură. Mai departe, am validat modelul prin implementarea unei platforme open source.

În cele din urmă, am abordat un nivel mai apropiat de hardware și am adresat problemele de securitate în Sistemele de Operare în Timp Real (RTOS). În această teză, ne-am folosit de avantajele privind securitatea pe care Rust le oferă. Rust e un limbaj de programare lansat relativ recent, dezvoltat pentru a permite scrierea de aplicații sigure. Datorită caracteristicilor sale, Rust e o alternativă viabilă la limbajul de programare C și poate fi folosit pentru scrierea de sisteme de operare. Astfel, a fost dezvoltat Tock, un sistem de operare pentru microcontrollere scris integral în Rust. Arhitectura Tock și faptul că e dezvoltat în Rust îi asigură un număr mare de politici de securitate implementate. Contribuția noastră se concentrează pe implementarea unui modul pentru Tock care permite procesarea de operații în timp real. Rezultatul este că Tock, un sistem de operare cu un nivel mare de securitate, poate fi folosit ca un RTOS.

Prin direcțiile de cercetare prezentate în această teză, am adresat principalele nivele software din infrastructurile Internet of Things. Am implementat multiple mecanisme de securitate și am adus contribuții menite să crească nivelul de securitate al procesului de dezvoltare în domeniul IoT.

Cuvinte cheie: Internet of Things, Securitate, Actualizări la distanță, Simulator hardware, Programare vizuală, Sistem de operare în timp real, Rust

1 INTRODUCERE

Fiind o tehnologie din ce în ce mai prezentă în viața oamenilor, Internet of Things (IoT) generează preocupări importante legate de siguranța infrastructurilor implementate, deoarece apariția unui atac de securitate poate avea un impact pe scară largă. De exemplu, celebrul atac Mirai Botnet a infectat peste 200.000 de dispozitive IoT [1, 2]. În acest context, securitatea dispozitivelor și a infrastructurii Internet of Things este esențială pentru integratorii comerciali și industriali.

Atunci când se abordează securitatea infrastructurilor IoT, o provocare importantă este reprezentată de complexitatea acestor sisteme. Arhitecturile Internet of Things sunt infrastructuri eterogene din punct de vedere hardware, software și al comunicațiilor. Ele se bazează pe o stivă complexă care începe cu dispozitive simple precum microcontrollere care rulează relativ puține linii de cod și se termină cu cloud-ul care agregă și procesează toate datele și gestionează dispozitivele conectate. Mai mult, în straturile inferioare, unde microcontrollerele și calculatoarele încorporate rulează aplicații, cea mai mare parte a software-ului este dezvoltată în C, ceea ce prezintă un risc ridicat de probleme de securitate. Industria afirmă că aproximativ 70% din vulnerabilități provin din probleme legate de siguranța memoriei [3, 4]. Acest lucru se datorează, în principal, modului în care C implementează operațiile de memorie care pot duce cu ușurință la vulnerabilități precum buffer overflow sau atacuri de tip flow-control [5]. În timp ce limbajele de programare moderne și mai sigure (ex. Python, Java) sunt din ce în ce mai mult utilizate pentru aplicații informatice integrate [6, 7], sistemele de operare utilizate în prezent pentru calculatoare și microcontrollere sunt scrise integral în C. Acest lucru rezultă în riscuri de securitate importante la nivelul kernel-ului. În plus, complexitatea și dimensiunile sistemelor de operare utilizate (ex. Linux are peste 27 de milioane de linii de cod [8]) expun o suprafață de atac mare.

Actualizările periodice sunt foarte importante, având în vedere importanța menținerii securității în infrastructurile IoT implementate în mediile comerciale și industriale. Acestea permit producătorilor să își mențină produsele la zi cu cele mai recente remedieri de erori și îmbunătățiri de securitate.

Toate provocările prezentate mai sus sunt abordate prin dezvoltarea de noi limbaje și tehnologii de programare.

Limbajul de programare Rust [9] aduce un progres semnificativ, fiind dezvoltat pentru a remedia deficiențele de securitate ale limbajului C. Acest limbaj de programare pentru sisteme este matematic garantat ca fiind sigur și implementează diverse măsuri de protecție pentru a preveni erorile legate de manipularea pointerilor. Acest lucru reduce numeroasele probleme de securitate întâlnite în mod obișnuit în programele C. Folosindu-se de avantajele Rust, multe

sisteme de operare scrise în întregime în acest limbaj sunt în curs de dezvoltare (ex. Tock [10], Hubris [11], Redox [12]). Dintre acestea, Tock, de exemplu, este un sistem de operare open-source destinat microcontrollerelor, fiind o alternativă adecvată la sisteme de operare precum FreeRTOS [13] sau Zephyr [14].

Mai mult, actualizările pentru dispozitivele IoT reprezintă, de asemenea, un domeniu intens cercetat, fiind dezvoltate mai multe soluții, dintre care multe cu scop comercial.

Prin urmare, definim următoarele întrebări de cercetare la care ne propunem să răspundem în această teză:

- Cum poate fi impusă securitatea sistemelor Internet of Things pe parcursul ciclului de viață al produsului, începând cu faza de prototipare și până la faza de întreținere?
- Sunt sistemele de actualizare la distantă utile în menținerea securității infrastructurilor IoT sau aduc riscuri suplimentare?
- În ce măsură putem securiza dispozitivele integrate cu capabilități reduse din infrastructurile IoT?
- Poate fi C înlocuit cu alte limbaje de programare moderne de nivel înalt care pot garanta o securitate sporită în microcontrollere și dispozitive integrate?

În contextul diverselor infrastructuri IoT care sunt implementate pentru diverse cazuri de utilizare, ne propunem să cercetăm modalități de securizare a acestor sisteme într-o manieră generică, indiferent de eterogenitatea care le caracterizează. În lucrarea noastră, abordăm mai multe nivele din stiva IoT și urmărim să propunem platforme sigure atât pentru kernel și spațiul utilizatorului, cât și pentru întreținerea acestor sisteme. Obiectivul final este de a realiza o infrastructură full-stack care să abordeze principalele probleme actuale de securitate în infrastructurile IoT.

Prin urmare, în dezvoltarea acestei teze, definim următoarele obiective intermediare legate de întrebările de cercetare și de contextul de securitate pe care le-am prezentat în secțiunea de mai sus:

- Analiza instrumentelor de dezvoltare și implementare existente pentru sistemele IoT pentru diferite cazuri și în diferite condiții de utilizare.
- Explorarea mecanismelelor de securitate ale sistemelor de actualizare la distantă a aplicațiilor și analiza eficienței acestora.
- Definirea constrângerilor pe care le au microcontrollerele și calculatoarele integrate și analiza modului în care aceste constrângeri afectează securitatea acestor dispozitive.
- Inspectarea securității sistemelor de operare existente pentru microcontrollere și calculatoare integrate.
- Identificarea alternativelor la sistemele de operare clasice, bazate pe C, pentru dispozitivele IoT.
- Definirea avantajelor pe care le au limbajele de programare de nivel înalt față de C din punct de vedere al securității.

- Identificarea alternativelor mai sigure la limbajul de programare C pentru aplicațiile Internet of Things.

În această teză, ne concentrăm pe securizarea stivei Internet of Things, propunând abordări generice și platforme independente de varietatea de dispozitive și tehnologii implicate. Scopul nostru este de a propune soluții adecvate pentru o mare varietate de infrastructuri care pot fi aplicate la mai multe nivele din stivă. Prin urmare, în activitatea noastră, abordăm securitatea tuturor componentelor majore ale infrastructurilor IoT. Începem cu aplicațiile IoT care rulează pe calculatoare integrate și sistemele de întreținere ale acestora implementate în cloud și coborâm în stivă până la securitatea microcontrolerelor. De asemenea, abordăm teme privind securizarea aplicațiilor și a sistemelor de operare pentru aceste dispozitive.

În prima contribuție, ne concentrăm asupra securității spațiului utilizator pentru gateway-urile IoT și calculatoarele integrate. Analizăm securitatea pe care limbajele de programare moderne o aduc la nivelul aplicației, deoarece majoritatea implică un sandbox în care este executat codul. În acest context, contribuția noastră se concentrează pe identificarea unora dintre limbajele de programare de nivel înalt care sunt potrivite pentru zona IoT și pentru dispozitivele încorporate specifice [15]. Propunem cazuri de utilizare pentru acestea și cercetăm cât de versatile sunt și cât de ușor pot fi adaptate pentru a rula pe o nouă arhitectură hardware. În acest fel, am identificat JavaScript, împreună motorul JerryScript [16], ca fiind unul dintre cele mai potrivite limbaje care pot fi utilizate pentru programarea unei mari varietăți de dispozitive integrate, de la cele mai puțin puternice la altele cu capacități mai avansate [17].

În continuare, am abordat aspecte legate de instrumentele de dezvoltare pentru diferite cazuri de utilizare. Definim cerințe specifice în funcție de cazul de utilizare a dezvoltării: comercial, prototip sau educațional și propunem soluții sigure pentru fiecare dintre ele. Mai departe în cercetarea noastră, propunem o soluție generică capabilă să abstractizeze operațiunile legate de configurarea hardware și alte configurații [18,19]. Testăm eficiența soluției în mai multe medii cu diverși utilizatori, propunând o platformă generică care permite integratorilor, pasionaților și studenților care se pregătesc să lucreze în acest domeniu să se concentreze pe aspecte precum securitatea infrastructurilor IoT pe care le dezvoltă [20, 21].

În același domeniu legat de instrumentele de implementare și întreținere, am abordat întreținerea aplicațiilor IoT, deoarece, odată ce un sistem IoT este implementat, la un moment dat trebuie efectuate diverse modificări (ex. remedieri de erori, îmbunătățiri de securitate, noi caracteristici). Aceste actualizări sunt efectuate de la distanță, ceea ce introduce multiple dificultăți și riscuri. Contribuția noastră în această teză constă într-o analiză detaliată a provocărilor legate de infrastructurile de instalare și actualizare a aplicațiilor, urmată de propunerea unui model matematic generic pentru o astfel de infrastructură care acoperă toate aspectele identificate. Mai mult, validăm modelul cu o implementare specifică a unui caz de utilizare [22].

Contribuția finală se concentrează asupra dispozitivelor de la baza stivei IoT: senzorii și actuatorii. Acestea se bazează pe microcontrollere cu consum redus de energie și memorie redusă, care nu pot rula software complex, cum ar fi un sistem de operare complet. În cazul aces-

tor dispozitive, securizarea lor reprezintă o provocare și mai mare din cauza constrângerilor hardware. Prin urmare, în această teză, începem cu o analiză aprofundată a riscurilor de securitate implicate în dezvoltarea acestor dispozitive și identificăm soluțiile existente [23]. În plus, propunem un sistem de operare în timp real, securizat pentru microcontrollere, care utilizează tehnologii și limbaje de programare avansate precum eBPF și Rust [24].

2 ENFORSAREA SECURITĂȚII PRIN LIMBAJE DE PROGRAMARE MODERNE

În acest capitol, ne concentrăm pe demonstrarea faptului că utilizarea pentru dispozitivele IoT a limbajelor de programare care rulează folosind un sandbox este posibilă și facilă. Prin urmare, ne propunem să identificăm domeniile în care C poate fi înlocuit cu un limbaj de programare care reduce riscul de erori și breșe de securitate.

2.1 Integrarea limbajului de programare D în dispozitivele IoT cu restricții

Limbajul de programare D este un limbaj de uz general menit să fie succesorul lui C++ [25]. Deși este similar cu C din punct de vedere al sintaxei, D își propune să fie o alternativă mai sigură.

Având în vedere că D este utilizat în sisteme de uz general și chiar în kernelul Linux [26], scopul cercetării noastre este de a testa dacă putem rula D pe dispozitive cu capacități reduse, cum ar fi microcontrollerele, pentru a înlocui C cu o alternativă mai sigură.

D este un limbaj compilat care se bazează pe DRuntime. DRuntime este biblioteca care definește limbajul D și datorită dimensunilor sale, este motivul pentru care nu putem rula aplicații D sigure pe microcontrollere. DRuntime are o dimensiune de aproximativ 40 MB care nu se potrivește cu constrângerile de memorie ale majorității MCU-urilor disponibile pe piață.

Prin urmare, în această cercetare, ne propunem să adaptăm mediul de programare D pentru a rula pe microcontrolere fără a compromite avantajele sale.

Am ales să implementăm cercetarea noastră pe un dispozitiv Nucleo F429ZI, care este un ARM Cortex-M4 cu o memorie flash de 2MB și SRAM de 256+4 KB. La nivelul sistemului de operare, am ales Tock, un sistem de operare open-source scris în Rust care are suport complet pentru acest dispozitiv.

Principalul aspect abordat este de a permite cross-compilarea aplicațiilor D bazate pe DRuntime pentru arhitecturile ARM. Pentru aceasta, am folosit *ldc2*, un compilator bazat pe LLVM.

Având în vedere complexitatea sarcinii, am urmat o abordare în doi pași pentru acest subiect de cercetare.

În primul rând, ne-am concentrat asupra compilării aplicațiilor D folosind *better C*. Better C

este o opțiune care permite programatorilor să construiască aplicații D într-un mod similar cu C, deoarece elimină orice dependență de DRuntime.

În cazul nostru, utilizarea opțiunii better C depinde de construirea unei biblioteci de spațiu utilizator care ne permite să rulăm programe D peste Tock. După ce am implementat biblioteca, am obținut un mediu în care aplicațiile D independente de better C pot fi compilate și rulate pe dispozitivul Nucleo. Aplicațiile pot utiliza biblioteca standard Tock și pot controla orice periferice existente. Cu toate acestea, în acest moment, securitatea unei astfel de aplicații este similară cu orice altă aplicație C. De aceea, este necesar următorul pas.

Următorul pas constă în adaptarea DRuntime astfel încât să necesite mai puține resurse de memorie și să poată fi utilizat pentru a rula aplicații sigure pe microcontrollere. Cu toate acestea, la momentul redactării acestei teze, tehnologia existentă nu ne permite să compilăm DRuntime pentru arhitectura dorită. În plus, operatorul de concatenare a șirurilor de caractere nu poate fi utilizat dacă compilăm DRuntime pentru Nucleo. Utilizarea acestuia duce la o eroare a kernelului din cauza accesului la memorie nealocată sau protejată.

Această cercetare concluzionează că, în forma sa actuală, D nu pare a fi un candidat potrivit pentru dezvoltarea de aplicații IoT sigure. Astfel, am decis să abordăm alte limbaje de programare mai mature, cum ar fi JavaScript.

2.2 Dezvoltarea de aplicașii IoT sigure folosind JavaScript

Întrucât la momentul redactării acestei teze, D este încă departe de a fi potrivit pentru aplicații integrate pentru microcontrollere, identificăm JavaScript ca fiind o opțiune potrivită pentru scrierea de aplicații integrate mai sigure decât cele scrise în C.

În mod similar cu D, JavaScript utilizează un garbage collector care se ocupă de toate alocările de memorie, reducând riscurile de securitate în aplicații. În plus, JavaScript se bazează pe un executor, care aduce un alt nivel de securitate, deoarece codul din spațiul utilizatorului nu este executat direct pe procesor.

Codul JavaScript poate fi rulat cu ușurință pe microcontrollere folosind JerryScript. JerryScript este un motor JavaScript dezvoltat special pentru dispozitive cu capacități reduse. Acesta poate rula pe dispozitive cu mai puțin de 64KB de RAM și mai puțin de 200KB de ROM [16].

Având în vedere popularitatea JavaScript în domeniul IoT și securitatea pe care o aduce, scopul nostru în cercetarea prezentată este de a testa cât de ușor este să implementăm un program JavaScript pe un dispozitiv reprezentativ pentru o anumită categorie.

Dispozitivul pe care l-am ales pentru implementarea noastră este NXP Rapid IoT Prototyping Kit. Acesta conține un microcontroller NXP Kinetis K64 120MHz pe 32 de biți, bazat pe Arm Cortex-M4 Core, un controler wireless NXP Kinetis KW41Z pentru conectivitate BLE,

Thread și Zigbee și mai mulți senzori integrați: calitatea aerului, temperatură, giroscop, accelerometru, magnetometru etc.

Am folosit Amazon FreeRTOS la nivelul sistemului de operare, deoarece este sistemul de operare care are cel mai bun suport pentru acest dispozitiv.

Oficial, kitul Rapid IoT Prototyping Kit suportă doar C ca limbaj de programare.

2.2.1 Executarea JerryScript pe NXP Rapid IoT Prototyping Kit

Pentru a porta JerryScript pe un nou dispozitiv hardware, trebuie mai întâi să îl compilăm pentru acea platformă, obținând o bibliotecă statică. Odată compilat, motorul este capabil să ruleze bytecode-ul JerryScript. API-ul motorului expune funcții care permit dezvoltatorilor să ruleze aplicații JavaScript și, mai important, să definească funcții native care pot fi apelate în interiorul acestor aplicații.

Pentru cazul nostru de utilizare, am folosit codul sursă Amazon FreeRTOS furnizat de NXP și am adăugat motorul JerryScript ca un serviciu diferit. Pentru a compila această versiune a RTOS, am folosit aplicația MCUXpresso de la NXP, care a generat un binar ce poate fi încărcat pe dispozitiv. După ce a fost instalat, dispozitivul a executat codul JavaScript.

2.2.2 Compilarea și integrarea JerryScript

În timp ce aplicațiile JerryScript pot fi rulate cu succes pe dispozitivul NXP, platforma MCUXpresso folosită nu este ușor de folosit. Prin urmare, în această implementare, urmărim, de asemenea, să automatizăm procesul de implementare a aplicației și să eliminăm dependența de platforma furnizată de NXP.

În primul rând, am utilizat extensia Hexiwear specifică acestui dispozitiv. Atunci când extensia este atașată, plăca intră în modul de depanare și deschide o conexiune serială. Prin urmare, datele pot fi transferate de la computer la dispozitiv prin intermediul unui cablu USB.

În acest caz, soluția noastră a fost să construim un script care preia codul JavaScript și îl trimite prin interfața serială. Pe partea dispozitivului, am adăugat un serviciu la Amazon FreeRTOS, care a fost configurat să citească informațiile care vin pe linia serială și să le transmită către JerryScript pentru a le executa.

Această soluție este stabilă și funcționează fără nicio întrerupere. Cu toate acestea, necesită utilizarea extensiei hardware, care este costisitoare și, de asemenea, face ca dispozitivul să fie mai puțin robust.

A doua soluție pe care am implementat-o a fost trimiterea codului sursă prin conexiunea Bluetooth. În acest caz, creăm un serviciu FreeRTOS diferit care primește datele de la managerul Bluetooth și transmite codul către JerryScript. În acest fel, Rapid IoT poate fi

programat fără hardware suplimentar. Mai mult, nu este necesară nicio conexiune fizică între computer și dispozitiv.

Deși este ușor de utilizat și accesibilă, această soluție nu este stabilă. Testele noastre au dovedit o rată de succes de 50% în ceea ce privește transmiterea informațiilor corecte către dispozitiv. În plus, rezultatele arată că managerul Bluetooth de pe dispozitive se blochează din cauza unei scurgeri de memorie, deoarece modulul nu este conceput pentru transferul unor cantități atât de mari de date.

În această abordare, am construit modulul Amazon FreeRTOS JerryScript astfel încât să ruleze un cod sursă de dimensiuni fixe. Având în vedere memoria totală disponibilă, am inițializat un buffer de 32KB cu valori de zero. După ce binarul este generat cu ajutorul platformei MCUXpresso, îl salvăm în folderul proiectului, la o cale fixă. La fiecare execuție, injectăm codul sursă JavaScript pe care dorim să îl executăm în buffer-ul cu valori zero din binar. Analizând binarul, am identificat adresa de memorie unde este stocat șirul de caractere care reprezintă codul JavaScript (Figura 1). De fiecare dată când dorim să rulăm o aplicație, suprascriem acel spațiu de memorie din binar. Această soluție este posibilă, deoarece MCUXpresso nu semnează digital binarul pe care îl generează, astfel încât îl putem modifica și apoi îl putem afișa pe dispozitiv.

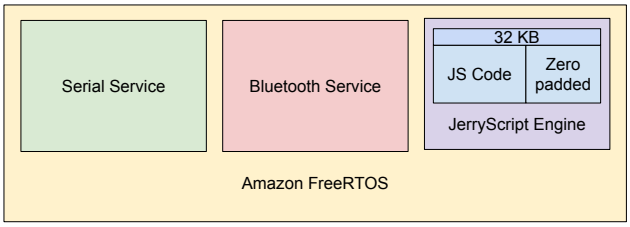

Figura 1: Arhitectura la nivel firmware.

Această abordare este cea mai stabilă. Codul JavaScript poate fi rulat cu succes pe dispozitiv în 100% din cazuri. În plus, în comparație cu abordările anterioare, în acest caz, pe măsură ce injectăm codul JavaScript, dispozitivul îl va rula chiar și după resetare. În celelalte două cazuri, acest lucru nu a fost posibil.

2.2.3 Teste și rezultate

Odată ce procesul de portare a avut succes, am atins obiectivul de a dovedi că JavaScript poate fi implementat cu ușurință pe microcontrollere. Mai departe, am testat această abordare de programare în comparație cu cea bazată pe C în diverse scenarii, iar prima s-a dovedit mai robustă și mai ușor de utilizat.

Rezultatul acestei cercetări este că JerryScript poate fi ușor de portat pentru diverse dispozitive hardware și considerăm că este o alternativă potrivită la C pentru dezvoltarea de aplicații mai sigure.

2.3 Programare bazată pe fluxuri pentru infrastructuri IoT

O altă abordare populară pentru programarea aplicațiilor IoT este utilizarea editorilor pe bază de flux, cum ar fi Node-RED [27]. Ideea principală din spatele acestei abordări este că aplicația poate fi considerată un flux de mesaje care declanșează acțiuni. Odată ce aplicația rulează pe dispozitiv, de fiecare dată când se declanșează unul dintre evenimentele specificate în nodurile utilizate, se generează un mesaj care este transmis către următorul nod. Receptorul va executa orice acțiune sau acțiuni de care are nevoie și va transmite același mesaj sau un mesaj nou.

Avantajul unei astfel de abordări este că dezvoltatorul poate vizualiza cu ușurință arhitectura sistemului și poate urmări modul în care interacționează fiecare dintre componente. Interfața vizuală facilitează depanarea, deoarece există o legătură clară între toate elementele.

Cu toate acestea, chiar dacă Node-RED are mai multe avantaje față de prima abordare, are totuși un mare dezavantaj: sincronizarea a două ramuri și asigurarea faptului că anumite funcții de pe diferite ramuri sunt apelate una după alta. Un alt dezavantaj este că fiecare nod poate emite mesaje în mod arbitrar, ceea ce duce la un comportament nedeterminist. Cu toate acestea, BPMN, o abordare bazată pe fluxuri utilizată în modelarea proceselor de afaceri, rezolvă aceste dezavantaje.

Soluția pe care o propunem în această secțiune este de a extinde BPMN pentru a construi o platformă de dezvoltare mai fiabilă pentru aplicațiile Internet of Things.

2.3.1 Implementarea platformei

Pentru a construi o platformă de implementare pentru aplicațiile IoT bazate pe BPMN, folosim un editor BPMN existent care poate fi ușor de integrat în orice aplicație. Bpmn.js este un set de instrumente de redare a diagramelor open-source. Îl folosim pentru interfața care permite utilizatorilor să construiască aplicațiile. Setul de instrumente returnează o structură XML care conține toate elementele. Prin urmare, primul pas în implementare este de a traduce structura XML în JSON. Mai departe, trebuie să creăm controllerul pentru întregul proiect și unul pentru fiecare element [28].

Odată ce structura JSON este pusă la punct, trebuie să generăm un token pentru a parcurge rețeaua și a activa fiecare element. Fiecare element consumă token-ul (token-urile) primit(e) de la conexiunea (conexiunile) de intrare și generează un nou token care este transmis mai departe.

În implementarea noastră, token-ul este o structură care conține mai multe câmpuri, cum ar fi deviceId, tokenId, timestamp sau signature. Următorul pas este definirea comportamentului fiecărui element (ex. task, poartă XOR, eveniment de pornire) pe care îl putem utiliza pentru modelarea aplicației.

2.3.2 Implementarea aplicației și rezultatele obținute

Pentru a testa faptul că abordarea bazată pe BPMN poate fi aplicată la proiectarea aplicațiilor Internet of Things și că elementele existente sunt suficiente pentru construirea unui sistem complex, am proiectat o aplicație pentru o mașină de cafea folosind editorul BPMN propus.

Aplicația pe care am proiectat-o trebuie să controleze o mașină de cafea dotată cu un senzor NFC și un ecran HDMI pentru interacțiunea cu utilizatorul. Senzorul NFC detectează când un utilizator plasează un recipient în fața aparatului, apoi cafeaua este distribuită. Dacă utilizatorul are o ceașcă specială cu un ID NFC unic, aparatul poate identifica id-ul și îi poate arăta utilizatorului câte cești de cafea a băut în ziua respectivă. În cazul în care aportul său de cofeină depășește o anumită limită, este afișat un mesaj de avertizare. Distribuitorul are următorul comportament: când pornește, toarnă cafea timp de 20 de secunde, apoi se oprește automat și afișează un mesaj prin care îi cere utilizatorului să scoată ceașca. În cazul în care utilizatorul îndepărtează ceașca mai devreme de 20 de secunde, distribuitorul se oprește imediat. Odată ce distribuitorul se oprește, se calculează cantitatea de cafea turnată.

Am construit acest sistem folosind programarea procedurală și programarea bazată pe fluxuri, așa cum se arată în figura 2.

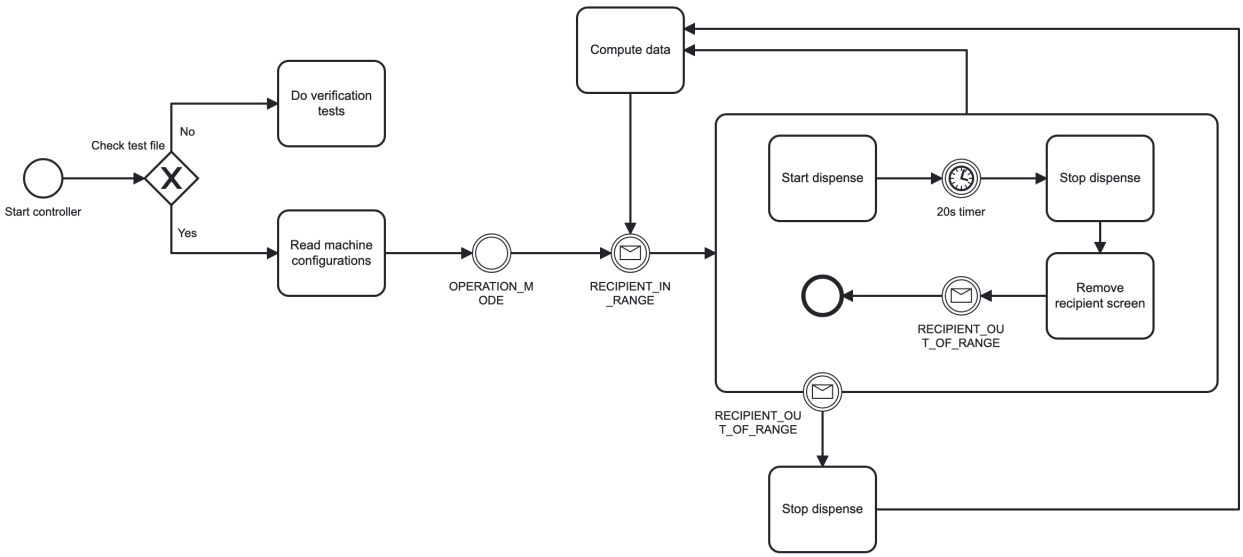

Figura 2: Aplicația IoT dezvoltată.

În cele din urmă, abordarea BPMN s-a dovedit mai ușor de depanat, reducând timpul de dezvoltare cu 30%. În plus, sistemul este mult mai ușor de urmărit și diagnosticat.

3 DEZVOLTAREA SECURIZATĂ A SISTEMELOR IOT

În acest capitol, propunem o platformă de dezvoltare pentru prototiparea aplicațiilor IoT securizate. De asemenea, abordăm provocările pe care le întâmpină educatorii în ceea ce privește predarea dezvoltării de infrastructuri IoT eficiente și sigure.

3.1 Instrumente de dezvoltare și prototipare IoT

La nivelul hardware, cel mai utilizat dispozitiv pentru prototiparea IoT este Raspberry Pi. Prin urmare, ne-am concentrat activitatea asupra construirii unei platforme de dezvoltare care face ca Raspberry Pi să fie ușor de programat și permite programatorilor să se concentreze pe dezvoltarea programelor, mai degrabă decât pe configurațiile necesare și pe instalarea aplicațiilor.

Wyliodrin, platforma pe care o propunem, este o aplicație web care utilizează protocoale securizate pentru a permite programarea și monitorizarea Raspberry Pi și a altor dispozitive similare într-o manieră simplă.

3.1.1 Prezentare generală a platformei

Platforma pe care o propunem, Wyliodrin, este o aplicație web care permite utilizatorilor să se conecteze de la distanță la dispozitivele lor integrate, precum Raspberry Pi, și să implementeze aplicații pe acestea. Platforma are următoarele caracteristici principale:

- Web - Este ușor de accesat și nu necesită instalarea niciunei aplicații pe computer.
- Configurare rapidă - Platforma ghidează utilizatorii printr-o serie simplă de pași necesari pentru a se conecta la dispozitiv. Nu sunt necesare cunoștințe avansate de rețea și programare pentru a controla dispozitivele.
- Monitorizare vizuală - Informațiile de depanare provenite de la dispozitive pot fi mapate pe grafice vizuale pentru o monitorizare mai ușoară.
- Consolă shell - Pentru utilizatorii mai avansați, este disponibilă o consolă shell. Aceasta permite un control complet asupra dispozitivului.
- Conexiune securizată - Deoarece securitatea este foarte importantă, toate datele pe care dispozitivele le schimbă cu aplicația web sunt criptate și trimise pe un canal securizat.
- Programare vizuală - Platforma are suport pentru un IDE bazat pe Google Blockly, care permite utilizatorilor să folosească blocuri pentru a crea aplicațiile dorite [29].

Aplicație Google Chrome

Platforma a fost inițial implementată ca o aplicație web care depinde de o conexiune la internet. Această abordare s-a dovedit a fi ineficientă în multe cazuri. În diverse configurații în care am utilizat platforma, rețeaua nu era suficient de fiabilă, iar dispozitivele se deconectau în mod constant. Prin urmare, am implementat o versiune mai stabilă, care funcționează și cu o conexiune de rețea locală. Mai mult, prin modificarea protocolului de comunicare, am crescut și fiabilitatea conexiunilor la distanță.

Pe măsură ce tehnologiile web mai avansate au devenit populare, am decis să rescriem platforma și să înlocuim complet supervizorul de dispozitive cu un altul scris în întregime în JavaScript (Node.js). În ceea ce privește aplicația web, am dezvoltat platforma sub forma unei aplicații Google Chrome, care poate fi instalată cu ușurință pe orice calculator. Am implementat, de asemenea, suport pentru conexiuni la distanță și o platformă web care poate rula din orice browser fără nicio instalare.

În acest caz, aplicația Chrome acționează ca server la care se conectează dispozitivul. Toate comunicările se realizează prin HTTP. Din punctul de vedere al utilizatorilor, acest lucru se traduce printr-o conexiune mai stabilă între dispozitiv și platforma web. Mai mult, am implementat și o modalitate de conectare de la distanță a dispozitivelor (prin internet) prin utilizarea unui server intermediar.

Arhitectura și implementarea platformei

Deoarece platforma a fost inițial dezvoltată pentru comunicarea XMPP și ulterior a trecut la HTTP, arhitectura a suferit, de asemenea, unele modificări. Pentru XMPP, singura abordare pe care am implementat-o este dependentă de un server XMPP terț. În schimb, atunci când se utilizează HTTP cu un dispozitiv aflat în aceeași rețea, aplicația Chrome acționează, de asemenea, ca server.

Serverul XMPP

Pentru a asigura un grad ridicat de securitate, sistemul inițial utilizează XMPP pentru mesajele schimbate între dispozitiv și browser. Principalele avantaje pe care le-am identificat în utilizarea XMPP sunt faptul că toată comunicarea este criptată, iar mesajele sunt trimise în format XML, care este ușor de extins.

În implementarea noastră, la un capăt se află utilizatorul care este conectat la platforma web. Apoi, aplicația web se conectează în numele utilizatorului la un server XMPP. Odată ce conexiunea este stabilită, serverul web recuperează listele de contacte, care sunt toate plăcile disponibile.

Serverul HTTP

Implementarea HTTP se bazează pe două tehnologii importante care permit transferul de date între dispozitiv și aplicația web:

- Avahi - un instrument de descoperire a dispozitivelor care permite aplicației web să identifice toate plăcile conectate la rețeaua locală;
- Web Sockets - un canal de comunicare bidirecțională care permite trecerea securizată a mesajelor între dispozitiv și aplicația web Wyliodrin.

```
Shell

Open
->
t: 's'
d: {a: 'o', c:columns, r: rows}

<-
t: 's'
d: {a: 'o', r:'d'}
or
t: 's'
d: {a: 'o', r: 'e', e: error}

Keys
<->
t: 's'
d: {a: 'k', t:keys}

<-
t: 's'
d: {a: 'e', e: error}
```

Figura 3: Reprezentarea mesajelor schimbate între dispozitiv și aplicația Wyliodrin.

Provocarea în implementarea acestui tip de comunicare a fost aceea de a permite aplicației web Wyliodrin să comunice cu mai multe dispozitive simultan. Acest lucru a necesitat un modul avansat pentru gestionarea instanțelor multiple de socket.

Supervizorul

Una dintre componentele esențiale ale platformei este supervizorul care rulează pe dispozitive. Acesta asigură comunicarea dintre server și dispozitiv. Pentru aceasta, pe baza modului în

care tehnologiile au evoluat în timp, am folosit diferite abordări de implementare.

Ca o prezentare generală, supervizorul este o aplicație care se conectează la server, primește comenzi sub formă de mesaje, le execută și trimite înapoi rezultatul. Codul sursă este, de asemenea, trimis sub formă de mesaj, iar un proces copil este inițiat pentru a-l compila și executa. În cadrul implementării inițiale, procesul copil rezultat comunica cu supervizorul prin intermediul unor pipe-uri. Cu toate acestea, pipe-ul nu suportă formatarea textului, astfel încât platforma putea afișa rezultatul într-o manieră neprietenoasă pentru utilizator. Prin urmare, am optat pentru crearea unui pseudoterminal (*pty*) pentru fiecare proces.

3.1.2 Întreținerea platformei

Un aspect esențial al platformei Wyliodrin este modul în care aceasta poate fi întreținută. Platformele precum Wyliodrin sunt dificil de actualizat, deoarece numărul de periferice crește continuu. Acest lucru înseamnă că trebuie să se lucreze continuu pentru a se asigura că sunt integrate cele mai recente dispozitive și senzori. Pe lângă configurarea conexiunilor necesare pentru fiecare nou dispozitiv suportat, elementele de programare vizuală trebuie, de asemenea, să fie actualizate. Fiecare producător de hardware care își creează senzorii oferă propriul mod de interacțiune cu perifericele respective. Astfel, numărul de blocuri vizuale necesare crește în același ritm cu numărul de periferice create.

De aceea, este nevoie de crearea unor astfel de blocuri într-un mod diferit de cel oferit de platforma Google Blockly. Modalitatea clasică presupune ca cineva să folosească platforma Google existentă pentru a crea fiecare bloc în parte și apoi să integreze elementul generat în platforma IoT care îl folosește. În acest context, ne concentrăm cercetarea asupra găsirii unei modalități de generare automată a blocurilor și de validare a faptului că aceasta este o abordare fezabilă.

Primul pas atunci când se integrează Blockly într-o platformă este de a crea un element web view care va conține editorul. Editorul este format dintr-un toolbox care enumeră toate blocurile și un dashboard în care utilizatorii pot plasa elementele. În afară de această configurare de bază, interfața poate fi personalizată.

Generarea automată a blocurilor pentru Google Blockly

Există mii de senzori și actuatori diferiți existenți care pot fi conectați la dispozitivele integrate existente. După cum am discutat în secțiunea anterioară, domeniul este într-o continuă expansiune și nu există încă un mod standard în care pot fi conectate perifericele. Acest lucru implică faptul că pentru fiecare grup de dispozitive de intrare/ieșire existente trebuie să se utilizeze funcții și protocoale de comunicare specifice.

Eclipse Mraa este o bibliotecă C/C++ creată de Eclipse IoT Project care oferă un API pentru a controla porturile de intrare/ieșire ale diferitelor dispozitive integrate [30]. Biblioteca are,

de asemenea, suport pentru Python și JavaScript, lucru care facilitează utilizarea ei.

Prin urmare, este mai ușor pentru producătorii de periferice să își mapeze senzorii, în timp ce utilizatorii văd un set generic de funcții pe care trebuie să le folosească pentru a controla hardware-ul. Prin utilizarea acestei biblioteci, numărul de funcții care trebuie implementate s-a redus dramatic. Wyliodrin utilizează această bibliotecă pentru a minimiza numărul de blocuri care trebuie implementate.

Această secțiune prezintă un nou mod prin care Wyliodrin poate genera blocuri vizuale noi și actualizate.

Generatorul constă într-un script care extrage toate fișierele *libmraa* din depozitul git și le analizează, generând blocuri corespunzătoare fiecărei funcții descrise. Scriptul generează atât elemente vizuale, cât și cod Python și JavaScript. Toate acestea sunt stocate într-o bază de date deținută de Wyliodrin. Blocurile sunt preluate automat din baza de date, astfel încât acest proces să fie transparent pentru utilizatorii Wyliodrin, care văd un toolbox complet și actualizat.

Arhitectura generatorului

Generatorul se bazează pe biblioteca *libmraa*, care ne-a adus la ideea că odată ce există câteva funcții standard generice, este ușor și eficient să creăm un generator automat de blocuri.

Primul pas a fost să extragem din fișierele bibliotecii informațiile necesare pentru a obține câteva elemente vizuale intuitive. Odată ce formatul fișierelor libmraa a fost clar, următorul pas a fost generarea analizatorului de fișiere. Pentru aceasta, am folosit un generator de parser, Jison [31]. Orice intrare va fi convertită în arborele abstract de sintaxă, care este, de fapt, o structură JSON [32].

După ce structura de bază este extrasă din fișierul sursă, următorul pas este analiza semantică, care verifică dacă structura JSON de bază este tradusă corect într-una care conține toate informațiile relevante pentru generarea unui bloc.

După ce totul este analizat, următorul pas este efectuarea analizei semantice. În primul rând, numele funcției trebuie să fie separat de tipul acesteia. De asemenea, în cazul în care un argument al funcției este de tip *enum*, acesta trebuie să fie legat de acea structură *enum*. După acest pas, sunt disponibile informațiile esențiale pentru generarea automată a blocurilor. Cu toate acestea, blocul rezultat nu va fi foarte diferit de funcția pe care o reprezintă.

Scopul Google Blockly este de a înlocui codul prin utilizarea unor elemente mai intuitive. De aceea, este important să se ajusteze această structură pentru a o face mai potrivită pentru acest limbaj de programare. Aceasta este o fază de optimizare și constă în crearea unui bloc bine construit. Ceea ce am făcut, este să interpolăm descrierea fiecărei funcții, care este practic textul principal de pe bloc, cu argumentul, acolo unde este posibil.

În final, rezultatul este o nouă structură JSON care conține toate informațiile relevante necesare pentru a genera corect un element Blockly.

Pornind de la toate aceste informații, toate celelalte detalii necesare pot fi deduse. De aceea, următorul pas este crearea unei structuri mai complexe din care elementele vizuale pot fi generate cu ușurință.

În timpul acestui proces, fiecare parametru este transformat într-o intrare împreună cu câmpurile corespunzătoare. De exemplu, orice parametru boolean este transformat într-o casetă de verificare, în timp ce orice enumerator se traduce într-o listă derulantă. În plus, tipurile de variabile de bază, cum ar fi *int* sau *char** sunt convertite în *Number*, respectiv *String*.

În final, rezultatul este o nouă structură JSON care reprezintă intuitiv corespondența pe care o are cu blocul creat.

Generator automat de blocuri pentru Wyliodrin

Pornind de la generatorul de blocuri prezentat în secțiunea anterioară, am creat un API care permite crearea, manipularea și stocarea fiecărei structuri de blocuri noi. API-ul are două părți, una este destinată utilizării de către serverul web pentru a genera blocuri și pentru a modifica structura JSON, în timp ce cealaltă parte este destinată utilizării într-un client web pentru a afișa blocul și a introduce codul acestuia în pagină.

Partea serverului API include generatorul, deoarece expune funcțiile care generează structura JSON din funcțiile C/C++. În plus, API-ul gestionează și stocarea blocurilor. Aceasta comunică cu o bază de date care stochează structurile de date și le modifică.

Pe de altă parte, API-ul client permite afișarea blocurilor într-un browser. Modulul expune o funcție care analizează structura JSON și apelează funcțiile Blockly care construiesc blocul. Practic, această funcție înlocuiește funcția de inițializare a Blockly.

Odată ce reprezentarea fiecărui bloc este completă, este necesar un script care să extragă toate fișierele din libmraa, să genereze structura JSON finală și să o stocheze într-o bază de date. Pentru a îndeplini aceste sarcini, am creat un script Node.js.

Atunci când am utilizat generatorul de blocuri cu biblioteca *mraa*, am generat un total de 150 de blocuri. 80% dintre acestea sunt blocuri intuitive care pot fi utilizate ca atare. Restul necesită efectuarea unor modificări personalizate.

3.1.3 Utilizarea platformei

Platforma Wyliodrin este disponibilă pe GitHub ca un proiect open-source [33]. De asemenea, poate fi descărcată direct ca aplicație compilată pentru toate platformele existente (Windows, Linux și MacOS) sau poate fi utilizată direct în browser [34].

Pentru a măsura utilizarea platformei, am integrat un modul care ne permite să numărăm numărul de proiecte construite și numărul de utilizatori noi. Am început să colectăm date de utilizare în octombrie 2019. În intervalul 1 octombie 2019 - 31 decembrie 2022, un număr total de 30.000 de utilizatori au dezvoltat aplicații utilizând versiunile web și locală. Mai mult, un total de 40.000 de aplicații noi au fost dezvoltate și implementate pe aproximativ 31.000 de dispozitive Raspberry Pi.

3.2 Educația în domeniul IoT

Deoarece proiectele educaționale IoT sunt, într-o anumită măsură, similare prototipurilor construite de comunitate, am adaptat platforma Wyliodrin pentru acest domeniu.

Deși unele dintre caracteristicile platformei standard Wyliodrin sunt potrivite pentru nevoile educaționale (ex. programarea vizuală, configurarea ușoară), este necesară dezvoltarea de caracteristici suplimentare. În acest scop, în cadrul cercetării noastre, am implementat următoarele instrumente hardware și software pentru simplifica procesul educațional:

- o extensie hardware pentru a abstractiza lucrul cu circuitele electronice;
- un simulator hardware pentru instituțiile care nu-și pot permite extensia propusă.

3.2.1 Platformă hardware pentru abstractizarea circuitelor

Platforma hardware propusă este un shield pe care studenții îl folosesc pentru a construi ușor circuitul pe care software-ul lor îl va controla.

În proiectarea platformei, am ținut cont de faptul că elevii înțeleg mai bine informațiile prezentate dacă le pot utiliza în aplicații din viața reală. Platforma constă atât în componente hardware, cât și software concepute pentru a fi implementate într-o clasă de elevi. Sistemul este construit pe baza unor instrumente educaționale existente, cum ar fi Blockly și Raspberry Pi [35].

Soluția noastră constă într-o extensie pentru Raspberry Pi care expune o gamă largă de conectori Grove la care elevii pot conecta periferice precum butoane, LED-uri, panouri solare, senzori de temperatură și lumină etc. Pentru a conecta perifericele, elevii nu au nevoie de cunoștințe de electronică, deoarece toate elementele sunt concepute pentru a fi conectate direct în porturi standard, fără a fi necesare alte conexiuni.

Dispozitivul hardware propus constă din următoarele elemente, așa cum se observă în figura 4:

- LCD;
- 3 LED-uri;

- 2 butoane;
- 30 de conectori Grove pentru conexiuni digitale, analogice și I2C.

Figura 4: Extensia pentru Raspberry Pi.

Pentru a simula mai bine sistemele pe care studenții le construiesc, pe lângă extensia Raspberry Pi, am construit o placă de susținere pe care aceștia pot plasa obiecte imprimate 3D, cum ar fi semafoare sau structuri de case. Am proiectat obiectele pentru a fi pe deplin compatibile cu perifericele Grove, astfel încât elevii să le poată atașa LED-uri și senzori, obținând un obiect fizic (Figura 5).

Pentru programarea propriu-zisă a dispozitivelor, elevii pot alege dintre limbaje de programare precum Python, JavaScript sau Blockly. În acest fel, elevii obișnuiți cu programarea pot folosi un limbaj procedural precum Python, în timp ce elevii fără cunoștințe de programare pot începe să utilizeze blocuri vizuale.

Folosirea în cadrul clasei

Placa de extensie Wyliodrin a fost utilizată în cadrul unui curs de programare pentru studenții de la facultatea de Energetică, pe parcursul unui semestru. Pentru a evalua eficiența platformei, am luat în considerare parametri precum implicarea studenților, complexitatea aplicațiilor construite de studenți, notele lor finale și progresul general al grupei. Am comparat acești parametri cu cei obținuți de grupa anterioară de studenți din anul precendent. În timp ce studenții din anul precedent au lucrat cu dispozitive simple Raspberry Pi și Arduino pentru

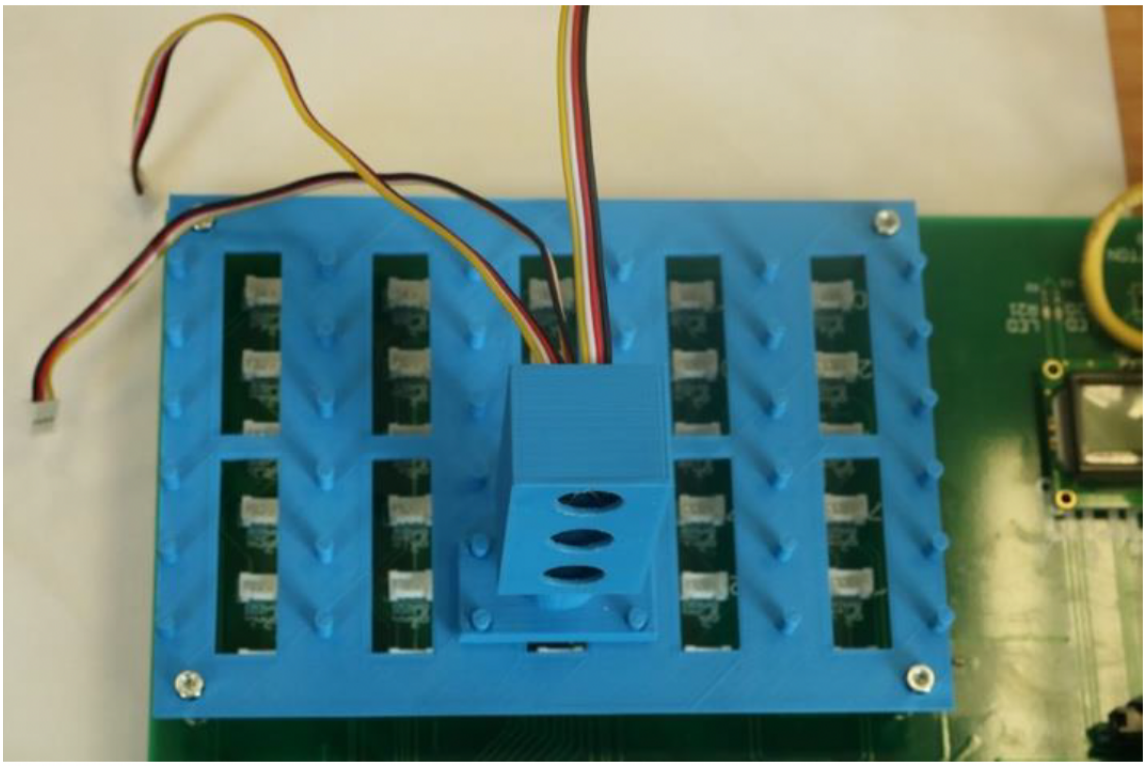

Figura 5: Semafor conectat la extensia pentru Raspberry Pi.

a controla periferice simple, studenții din anul următor au folosit platforma Wyliodrin Lab pentru aplicații legate de domeniul de energetică [36, 37].

În anul folosirii sitemului Wyliodrin, laboratoarele au constat în cerințe care includ construirea unei centrale energetice, controlul și monitorizarea energiei consumată de clădiri, semafoare și sisteme de iluminat stradal [38]. Rezultatul a fost că studenții s-au concentrat asupra laboratorului și au rezolvat toate exercițiile solicitate, demonstrând un nivel ridicat de implicare. În schimb, studenții sin anul anterior obișnuiau să își piardă interesul și să nu rezolve toate exercițiile de laborator.

O altă abordare a fost aceea de a le prezenta studenților programarea vizuală bazată pe Blockly și de a-i ajuta să facă tranziția către programarea Python. Pentru a face acest lucru, le-am cerut studenților să construiască o aplicație completă folosind blocurile și apoi să modifice aplicația din codul Python [39]. După câteva laboratoare, am creat câteva elemente Blockly care generează un cod Python defect și i-am rugat pe elevi să repare codul. În cele din urmă, le-am cerut elevilor să construiască aplicații Python simple. Până la sfârșitul semestrului, 80% dintre studenți au fost capabili să scrie o aplicație Python care citește date de la senzori și controlează câteva LED-uri.

Pentru examenul final, 70% dintre studenți au reușit să controleze periferice, să facă schimb de date între diferite dispozitive Raspberry Pi și să utilizeze servicii web. În schimb, la sfârșitul semestrului din anul precedent, majoritatea studenților au avut cunoștințele de bază pentru a controla dispozitivele Raspberry Pi și Arduino, foarte puțini studenți având abilități mai

avansate.

3.2.2 Platformă de simulare hardware

Implementarea cursurilor de IoT implică costuri financiare semnificative. Acest lucru se datorează necesității de componente hardware, începând cu placa de dezvoltare principală, unde Raspberry Pi și Arduino sunt cele mai populare alegeri, până la senzori, actuatori și alte componente suplimentare [40]. Un kit care conține un Raspberry Pi, un breadboard și un set de periferice de bază are un cost mediu de 100 de dolari, ceea ce duce la un cost total de echipare a unui laborator IoT de aproximativ 2000 de dolari.

În contribuția detaliată în această secțiune, ne propunem să prezentăm o soluție pentru problema prezentată mai sus. Pentru a reduce costurile legate de implementarea laboratoarelor IoT, am proiectat un simulator software conceput special pentru educație. Platforma pe care o propunem simulează un dispozitiv Raspberry Pi conectat la diverse periferice, care poate rula aplicații de bază. Această soluție urmărește să facă din IoT un domeniu mai accesibil și ajută la introducerea unor astfel de cursuri în instituțiile de învățământ prin reducerea costurilor.

Pentru a realiza acest lucru, am adăugat un simulator Raspberry Pi ca un modul suplimentar la platforma Wyliodrin. Principalul avantaj este că studenții pot utiliza aceeași platformă pentru a rula aplicații atât în simulator, cât și pe un Raspberry Pi fizic. Mai mult, nu sunt necesare modificări pentru a rula o aplicație pe un Raspberry Pi fizic sau un Raspberry Pi simulat.

Simulatorul Raspberry Pi funcționează pe baza schemelor Fritzing care pot fi importate în aplicații. În prezent, simulatorul recunoaște următoarele periferice: LED-uri, LCD-uri și butoane. Prin urmare, utilizatorii pot construi orice schemă care utilizează aceste componente și le pot importa în platforma pe care o propunem.

Mai departe, pe măsură ce un circuit este selectat, studenții pot scrie aplicații și le pot rula pe dispozitivul simulat. În prezent, aplicațiile pot fi scrise fie în Node.js, fie folosind Blockly, care generează cod JavaScript.

Utilizarea în cadrul laboratoarelor

Pentru a testa eficiența soluției, am folosit-o într-un curs de IoT pentru studenții din primul an de inginerie electrică, fără cunoștințe anterioare de lucru cu Raspberry Pi. Aproximativ 25 de studenți au participat la acest curs, care a durat 12 săptămâni, constând într-un curs teoretic de două ore și un laborator de două ore.

În primele patru săptămâni, studenții au folosit doar simulatorul Raspberry Pi integrat în Wyliodrin și au trecut apoi la lucrul cu dispozitivul fizic. Rezultatul a fost că studenții au reușit să implementeze circuitele simulate și apoi au trecut la implementarea schemelor fizice

fără să compromită niciun component hardware. În comparație, cu un an în urmă, când soluția nu a fost disponibilă și studenții au început prin a implementa direct circuite pe Raspberry Pi, ei obișnuiau să conecteze prost LED-urile, stricând multe dintre ele din cauza scurtcircuitelor. Mai mult, studenții au reușit să dezvolte aplicații mai avansate până la sfârșitul semestrului.

4 INFRASTRUCTURĂ DE ACTUALIZARE SECURIZATĂ PENTRU SISTEMELE IOT

Scopul cercetării prezentate în acest capitol este de a propune o infrastructură pentru implementarea și actualizarea software-ului, construită pe un model generic, open-source și ușor de implementat de către orice integrator IoT.

4.1 Constrângerile platformei

În definirea infrastructurii, am analizat mai întâi contextul în care este utilizată și constrângerile existente. Ca urmare, am definit următoarele constrângeri de proiectare:

- *C1* - Platforma trebuie să permită dezvoltarea și depanarea aplicațiilor pe dispozitive hardware în condiții similare cu cele din producție.
- *C2* - Platforma trebuie să permită testarea beta a software-ului care urmează să fie implementat. În acest scop, software-ul este implementat pe același hardware ca și cel folosit în producție și în condiții similare cu cele din producție.
- *C3* - După ce testarea beta este finalizată cu succes, actualizările de software trebuie să fie implementate în mod incremental. În acest fel, orice eroare în procesul de implementare poate fi identificată din timp.
- *C4* - Actualizările aplicațiilor trebuie să fie programate în afara orelor de funcționare a dispozitivelor (de exemplu, nu putem actualiza o linie de asamblare în timp ce este în funcțiune sau un automat de vânzare în timp ce vinde produse).
- *C5* - În cazul în care aplicația nou implementată nu pornește pe anumite dispozitive, aceasta trebuie să fie readusă automat la cea mai recentă versiune funcțională.
- *C6* - Este necesar un tablou de bord de monitorizare pentru dispozitivele implementate. Responsabilii de întreținere vor avea acces la tabloul de bord pentru a vizualiza comportamentul dispozitivelor implementate, pentru a rula teste de diagnoză și pentru a accesa de la distanță dispozitivul pentru a efectua o restaurare manuală a software-ului, dacă este necesar.
- *C7* - În cazul unui dispozitiv compromis (ex. o bicicletă conectată furată), responsabilii cu întreținerea au posibilitatea de a-l dezactiva, aducându-l astfel într-o stare de nefuncționare.
- *C8* - Pentru securitate, dispozitivele trebuie să se autentifice și să fie compatibile cu TPM [41] (de exemplu, ARM TrustZone [42], Software Guard Extensions [43]).
- *C9* - Actualizările trebuie să fie rapide și să necesite transferuri mici de date.
- *C10* - Sistemul trebuie să fie deschis și ușor de integrat cu infrastructura furnizorului.

4.2 Modelul matematic propus

Pe baza constrângerilor definite mai sus, definim un model matematic pentru o infrastructură de implementare a software-ului integrat, care poate fi implementată de către furnizori folosind tehnologiile lor preferate.

4.2.1 Terminologie

Modelul pe care îl propunem se bazează pe o unitate centrală de orchestrare care gestionează dispozitivele, conexiunile acestora și procedurile de implementare. Dintr-o perspectivă mai detaliată, sistemul este format din următoarele elemente principale: *Furnizor, Utilizator, Produs, Cluster, Aplicație, Instalări de containere, Proiect, Eveniment*. În continuare, vom descrie în detaliu fiecare dintre aceste elemente:

- *Furnizor* - Este utilizatorul sistemului sau, mai precis, producătorul IoT, care construiește și gestionează dispozitivele. Furnizorul deține produse, clustere, proiecte, aplicații, instalări și containere.
- *Utilizator* - Pentru fiecare furnizor, pot fi creați mai mulți utilizatori. Fiecare utilizator poate gestiona elementele descrise mai departe.
- *Produs* - Fiecare dispozitiv înregistrat în platformă este un produs unic. Ne referim la acesta ca la un produs, deoarece este elementul de bază pe care îl vinde firnizorul. Fiecare produs se identifică în mod unic printr-un id și are, de asemenea, un nume. Definim trei tipuri diferite de produse: dezvoltare (utilizat pentru dezvoltarea și depanarea software-ului), beta (utilizat pentru testarea beta) și producție (comercializat de către vânzător). Fiecare produs face parte dintr-un cluster (definit mai jos).
- *Cluster* - Un cluster poate conține unul sau mai multe produse. Toate produsele dintr-un cluster sunt localizate într-o zonă geografică delimitată și rulează același software (de exemplu, o colecție de dispozitive care colectează date despre sol într-o anumită zonă agricolă).
- *Aplicație* - O aplicație definește în mod unic un software care urmează să fie instalat pe un cluster. Fiecare aplicație are o listă de versiuni. Mai mult, pentru fiecare aplicație, este definită o listă de parametri impliciți utilizați pentru a rula software-ul.
- *Container* - Este software-ul efectiv care urmează să fie instalat pe produsele înregistrate. Containerul este stocat într-un repository.
- *Instalare* - O instalare leagă unul sau mai multe clustere de o aplicație, un număr de versiune a aplicației și un set de parametri ai aplicației. Odată ce este creată o instalare, toate produsele din clusterul (clusterele) țintă vor rula versiunea specificată a aplicației selectate, utilizând parametrii definiți.
- *Eveniment* - Evenimentele sunt definiții ale acțiunilor care au loc în timpul funcționării sistemului (ex. înregistrarea produsului, crearea clusterului, o nouă versiune a aplicației).

Infrastructura de instalare a aplicațiilor pe care o modelăm în această secțiune este formată din trei componente de bază: serverul, produsele și instalările. Serverul comunică cu produsele, în timp ce produsele rulează software-ul livrat sub forma unei instalări. Pe baza acestei abordări generice, pot fi implementate cazuri de utilizare mai specifice pentru a optimiza sistemul de instalare și actualizări pentru scenarii specifice.

4.2.2 Mulțimi

Pentru a defini matematic infrastructura descrisă mai sus (interacțiunea dintre server, produse și instalări), definim următoarele mulțimi, pe baza cărora sunt implementate toate definițiile următoare:

- P - mulțimea care conține toate id-urile posibile ale produselor;
- C - mulțimea care conține toate id-urile posibile ale clusterelor;
- K - mulțimea care conține toate cheile posibile PKI (de exemplu, ECC [44] sau RSA [45]);
- P_a - mulțimea care conține toți parametrii posibili asociați cu instalările;
- A - mulțimea care conține toate id-urile posibile ale aplicațiilor;
- S - mulțimea care conține toate semnăturile digitale posibile bazate pe cheile din K;
- U - mulțimea care conține toate token-urile asociate unui produs; token-urile sunt utilizate pentru autentificarea produsului;
- E. - mulțimea care conține toate erorile definite;
- T - mulțimea care conține toate tipurile de produse posibile: *dezvoltare*, *beta*, *producție*.

4.2.3 Modelul matematic

.

Pe baza mulțimilor definite mai sus, definim următorul model pentru o platformă generică de implementare și actualizare a software-ului care respectă constrângerile definite anterior.

Definim mulțimea T care conține toate tipurile de produse disponibile (1).

$$T = \{development, beta, production\} \tag{1}$$

Definim M ca fiind spațiul tuturor produselor (2).

$$M = P \times K \times C \times K \times T \times P_a \tag{2}$$

Proiecția pe spațiul M, reprezentată de vectorul $\overrightarrow{m}$ are următoarele *dimensiuni*: id, cheie privată, id de cluster, cheie privată de cluster, tip și parametri suplimentari care pot fi utili pentru instalări specifice (3).

$$\overrightarrow{m} : M = (id_{produs}, key_{produs}, id_{cluster}, key_{cluster}, tip, parametru) \quad (3)$$

Pornind de la spațiul vectorial *M*, putem defini următoarele funcții:

- $produs$ (4) - funcția care proiectează $\overrightarrow{d}reaptam$ pe id_{produs} din *P*; rezultatul este o valoare care definește un produs;

$$product(\overrightarrow{d}reaptam) : M \to P = \overrightarrow{m} \times (1,0,0,0,0,0,0,0)^T \quad (4)$$

- k_p (5) - funcția care proiectează $\overrightarrow{m}$ pe $key_{product}$ din *K*; rezultatul este o valoare care definește cheia privată a produsului;

$$k_p(\overrightarrow{d}reaptam) : M \to K = \overrightarrow{m} \times (0,1,0,0,0,0,0,0)^T \quad (5)$$

- $cluster$ (6) - funcția care proiectează $\overrightarrow{m}$ pe $id_{cluster}$ din *C*; rezultatul este o valoare care definește clusterul care conține produsul;

$$cluster(\overrightarrow{d}reaptam) : M \to C = \overrightarrow{m} \times (0,0,1,0,0,0,0,0)^T \quad (6)$$

overrightarrow
- k_c (7) - funcția care proiectează $\overrightarrow{m}$ pe $key_{cluster}$ din *K*; rezultatul este o valoare care definește cheia privată a clusterului;

$$k_c(\overrightarrow{d}reaptam) : M \to K = \overrightarrow{m} \times (0,0,0,0,1,0,0,0)^T \quad (7)$$

- $type$ (8) - funcția care proiectează $\overrightarrow{m}$ pe $type$ din *T*; rezultatul este o valoare care definește tipul de produs;

$$tip(\overrightarrow{d}reaptam) : M \to T = \overrightarrow{m} \times (0,0,0,0,0,1,0)^T \quad (8)$$

- $parameters$ (9) - funcția care proiectează $\overrightarrow{m}$ pe $parameters$ din P_a; rezultatul este mulțimea de parametri necesar pentru fiecare implementare specifică;

$$parameters(\overrightarrow{m}) : M \to P_a = \overrightarrow{m} \times (0,0,0,0,0,0,1)^T \quad (9)$$

În legătură cu $k \in K$, definim k^T ca fiind cheia publică corespunzătoare.

4.3 Implementarea și validarea modelului - IoTWay

Pentru a verifica dacă modelul propus este potrivit pentru implementare, am construit o platformă de actualizări bazată pe acesta. Platforma se numește IoTWay [46] și este o soluție open-source bazată pe standarde și protocoale sigure. Cu toate acestea, pot fi realizate și alte implementări pornind de la modelul pe care l-am definit în secțiunea anterioară.

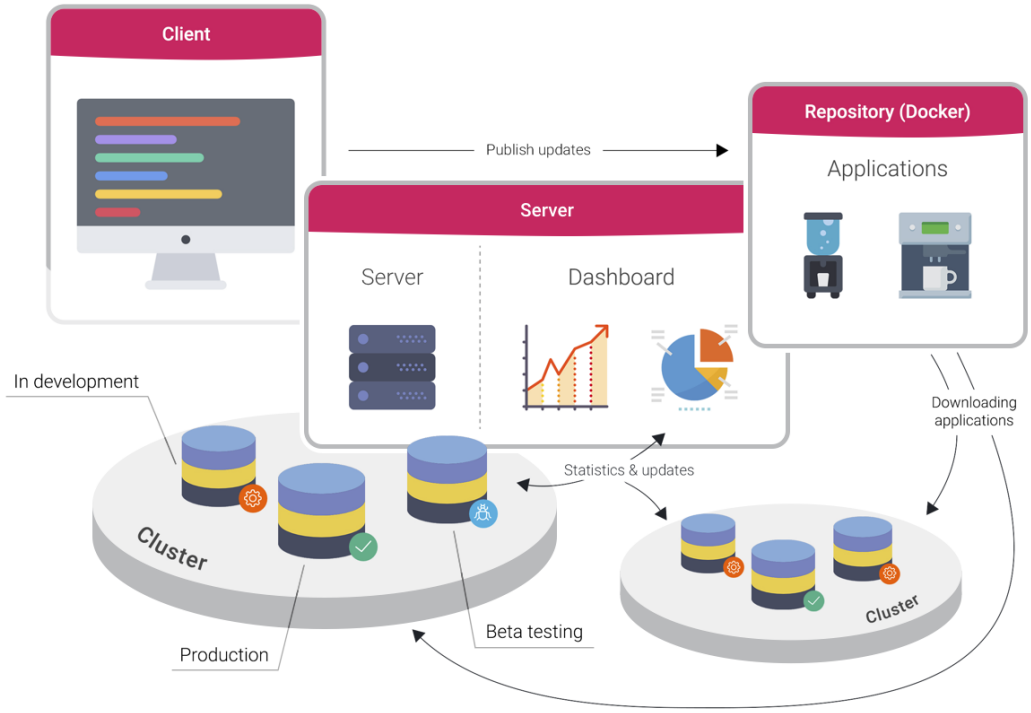

Figura 6: Arhitectura de nivel înalt a sistemului IoTWay.

Implementarea platformei se bazează pe cinci componente majore (server, repository, implementator, client, IDE) care sunt conectate așa cum este descris în figura 6.

Serverul este unitatea centrală, deoarece gestionează toate operațiunile esențiale legate de utilizatori, produse, clustere, aplicații și instalări. Conform *C10*, serverul este proiectat astfel încât platforma să poată fi integrată cu ușurință în infrastructura existentă. Serverul este construit ca o colecție de servicii web ce folosesc formatul JSON [47].

Repository-ul este un repository de containere care utilizează un token OAuth pentru autentificare [48].

Instalatorul este o aplicație care rulează pe fiecare produs și este responsabilă de modul în care containerele sunt instalate și pornite pe dispozitiv.

Clientul este o componentă opțională care oferă o interfață vizuală utilizatorilor. Prin intermediul clientului, vânzătorii pot gestiona toate operațiunile disponibile. Cu toate acestea, serverul poate fi integrat cu ușurință cu orice altă infrastructură utilizând API REST, iar clientul ar putea să nu fie necesar în acest caz.

IDE-ul este o altă componentă opțională. Este o aplicație web care permite furnizorilor să își creeze aplicațiile de la distanță. Este practic o interfață web în care utilizatorii își pot construi proiectele.

4.4 Studiu de caz

Pentru a evalua eficiența platformei IoTWay, am utilizat-o într-un proiect comercial. Cazul de utilizare a vizat implementarea unei infrastructuri de implementare și actualizare a software-ului pentru distribuitoarele inteligente de sucuri.

În acest studiu de caz, am conectat numeroase aparate de distribuție de sucuri care afișează reclame. Cu ajutorul IoTWay, furnizorul poate încărca un nou software pe dispozitiv și poate monitoriza aparatele implementate.

4.5 Rezultate

Pentru a măsura eficiența modelului și a instalării propuse, am implementat aproximativ 100 de automate de distribuție împreună cu un partener comercial, în trei regiuni diferite: România, SUA și India.

4.5.1 Caracteristicile software-ului

Platforma IoTWay a fost utilizată atât pentru dezvoltarea aplicației, cât și pentru actualizări. În acest context, software-ul implementat pe dispozitive variază de la aplicații simple până la aplicații mai avansate care implică o interfață cu utilizatorul.

Tabelul 1 prezintă comparația performanțelor între abordări. Media de încărcare a mașinii este calculată ca fiind media procentuală a utilizării CPU pe o perioadă de 10 minute.

Tabel 1: Performanța dispozitivului

Platforma	**CPU**	**RAM**	**Încărcare totală**	**Încărcare RAM**
BeagleBone Black	1.0 GHz	512 MB	150%	100%
Raspberry Pi 3 (no GPU driver)	4 x 1.2 GHz	1 GB	40%	60%
Raspberry Pi 3 (GPU driver)	4 x 1.2 GHz	1 GB	10%	60%

Deoarece dispozitivele BeagleBone Black au fost puternic încărcate, traficul de rețea a suferit multe pierderi de pachete și multe deconectări.

4.5.2 Performanța instalărilor

Primul parametru pe care l-am măsurat legat de eficiența instalării este dimensiunea software-ului pentru prima instalare și pentru actualizări. Deoarece IoTWay este conceput pentru a suporta actualizări diferențiate, se așteaptă ca prima instalare a software-ului să aibă o dimensiune semnificativ mai mare și să dureze mai mult timp.

Prima imagine pe care am creat-o a avut o dimensiune de 1,2 GB și a fost nevoie de 1h pentru ca implementarea să fie finalizată. Următoarele actualizări au o dimensiune cuprinsă între 200-300 MB și au avut nevoie de 10-15 min pentru a fi realizate. Având în vedere dimensiunile semnificative, 20% dintre actualizări au eșuat și au necesitat o nouă încercare. Cu toate acestea, niciuna dintre instalări nu a dus la reveniri sau la dispozitive corupte.

După primele măsurători, am redus dimensiunea imaginii containerului ceea ce a dus la o dimensiune inițială a imaginii containerului de 500 MB și la containere de actualizare care variază între 50 MB și 100 MB. Astfel, timpul de implementare inițială a scăzut la 20-30 de minute, iar timpul de actualizare la aproximativ 5 minute.

Tabel 2: Performața actualizărilor.

	Instalarea inițială	**Actualizare**	**Reincercări**	**Timp instalare inițială**	**Timp actualizare**
Aplicația inițială	1.5 GB	500 MB	20%	1 h	10-15 min
Aplicația optimizată	200-300 MB	50-100 MB	5%	20-35 min	5 min

Inițial, măsurătorile au fost efectuate pe Raspberry Pi (Tabelul 2). În schimb, BeagleBone Black are constrângeri hardware care afectează infrastructura de instalare, ceea ce ne-a condus la o frecvență și un timp de reluare a actualizărilor mai mari cu aproximativ 30%.

După optimizarea containerelor, am utilizat infrastructura pentru a efectua un total de 133 de instalări pe 100 de mașini de distribuție (Tabelul 3). Pentru dispozitivele BeagleBone Black, 30% dintre instalări au eșuat din cauza pierderilor de pachete de rețea și a scrierilor defectuoase pe disc. În toate aceste cazuri, sistemul a fost readus cu succes la cea mai recentă versiune funcțională a aplicației. Am avut, de asemenea, 20 de aparate cu eșecuri complete din cauza cardurilor SD corupte.

Tabel 3: Statistici ale actualizărilor.

Platforma	**Produse**	**Actualizări**	**Media Dispozitive recuperate**	**Media Dispozitive nerecupăerate**	**Dispozitive cu alte erori**
BeagleBone Black	80	133	25	2	20
Raspberry Pi 3	30	133	3	0	0

4.5.3 Comparare performanțelor cu Balena

Cercetările prezentate în acest capitol au ca scop principal să propună o infrastructură de implementare și actualizare de software care să aibă o bază teoretică solidă și care să poată fi ușor adaptată pentru diverse cazuri de utilizare.

Atunci când se compară modelul și implementarea propusă cu alte platforme, Balena se evidențiază ca o infrastructură de actualizare care are mai multe mecanisme comune cu platforma IoTWay. Am implementat cazul de utilizare prezentat mai sus utilizând infrastructura Balena. Rezultatele sunt prezentate în tabelul 4.

Tabel 4: Comparația performanțelor sistemelor de instalare IoTWay și Balena.

Platforma	**Produse**	**Actualizări**	**Medie recuperate**	**Medie nerecuperate**	**Produse cu erori**
IoTWay	110	133	19	2	20
Balena	110	133	17	10	40

5 SECURITATEA LA NIVELUL KERNELUI

Acest capitol propune o componentă de timp real pentru un sistem de operare încorporat care asigură un grad ridicat de securitate. După o scurtă trecere în revistă a tehnologiilor existente dedicate aplicațiilor integrate, concluzionăm că toate RTOS-urile clasice prezintă o penalizare majoră în materie de securitate, strâns legată de faptul că toate aceste sisteme de operare sunt scrise în C. Datorită modului în care este implementată gestionarea memoriei în C, sunt frecvente accesele incorecte de memorie (buffer overflow), precum și alte atacuri similare.

Pe de altă parte, Hubris și Tock, sisteme de operare integrate scrise în întregime în Rust, nu sunt afectate de majoritatea riscurilor de securitate legate de gestionarea memoriei în C. Unul dintre aspectele care le lipsește este suportul de operații în timp real. Prin urmare, ne-am propus să îmbunătățim Tock și să integrăm un mecanism astfel de mecanism pentru a obține un nivel de securitate superior, asigurând în același timp tratarea rapidă a evenimentelor specifice.

5.1 Aplicarea securității în Tock

Siguranța sistemului de operare Tock se bazează pe trei caracteristici principale de implementare:

1. Kernelul este scris în Rust, numărul de linii de cod *unsafe* fiind redus la minimum.
2. Driverele sunt împărțite în două nivele: driverele de nivel scăzut cu acces direct la hardware și capsule, driverele de nivel superior care fac abstracție de cele de nivel scăzut și nu au voie să folosească *unsafe* Rust. Cea mai mare parte a dezvoltării se realizează la nivelul capsulelor.
3. Folosește protecția hardware a memoriei pentru a restricționa accesul aplicațiilor la memoria din afara spațiului lor de adrese.

Aplicațiile rulează în spațiul utilizator și sunt compilate complet separat de kernel. Avantajul acestei abordări constă în faptul că implementarea kernel-ului și a aplicațiilor poate fi realizată separat, iar aplicațiile sunt instalate într-un mod similar cu cele pentru sistemele de operare de uz general.

Figura 7 descrie stiva de implementare Tock, care demonstrează distincția dintre kernel și spațiul utilizator [49].

În timp ce această separare clară între kernel și spațiul utilizator are avantaje de siguranță, ea introduce întârzieri în procesul de gestionare a întreruperilor, deoarece aplicațiile nu pot

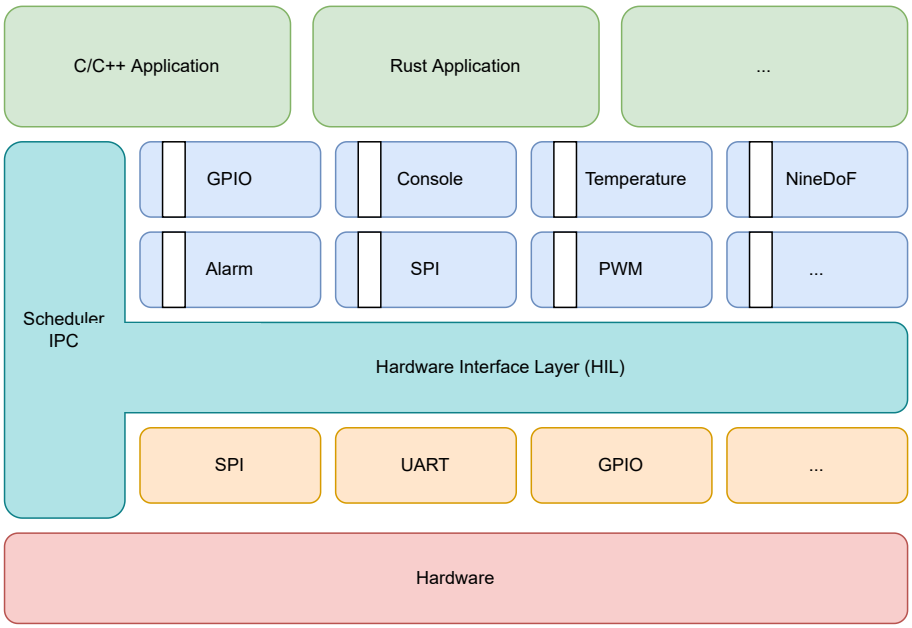

Figura 7: Stiva Tock.

executa cod direct în timpul întreruperilor.

5.2 Abordarea Propusă

Luând în considerare stiva Tock, unul dintre costurile suplimentare pe care le-am identificat ca împiedicând sistemul de operare să execute operațiuni în timp real cu latență redusă este schimbarea contextului de la kernel la spațiul aplicației. Latența este generată în principal de decalajul de timp dintre momentul în care kernel-ul observă întreruperea și momentul în care se execută codul funcției de callback din spațiul utilizator.

Considerăm că mutarea întregii gestionări a întreruperilor din spațiul utilizator în kernel și eliminarea schimbării de context asociate cu executarea unei rutine de întrerupere va reduce semnificativ timpul de răspuns.

Am ales să folosim filtrul de pachete Berkeley (Berkeley Packet Filter - BPF) pentru a injecta cod specific în kernel Linux în timpul execuției [50]. Este o modalitate simplă de a introduce în mod dinamic cod pentru kernel din spațiul utilizator fără a recompila kernel-ul.

5.3 Integrarea Sandbox-ului eBPF în Tock

Implementarea pe care o propunem constă în injectarea rutinei de gestionare a întreruperilor în kernel (driver) în loc de înregistrarea funcției de callback pentru upcall, așa cum se arată în figura 8.

Rutina injectată este orice bytecode eBPF pe care dezvoltatorul aplicației îl poate genera

folosind diverse instrumente, cum ar fi LLVM sau gcc. Deoarece eBPF este un cod portabil standard cunoscut, această abordare asigură faptul că soluția nu depinde de o anumită arhitectură.

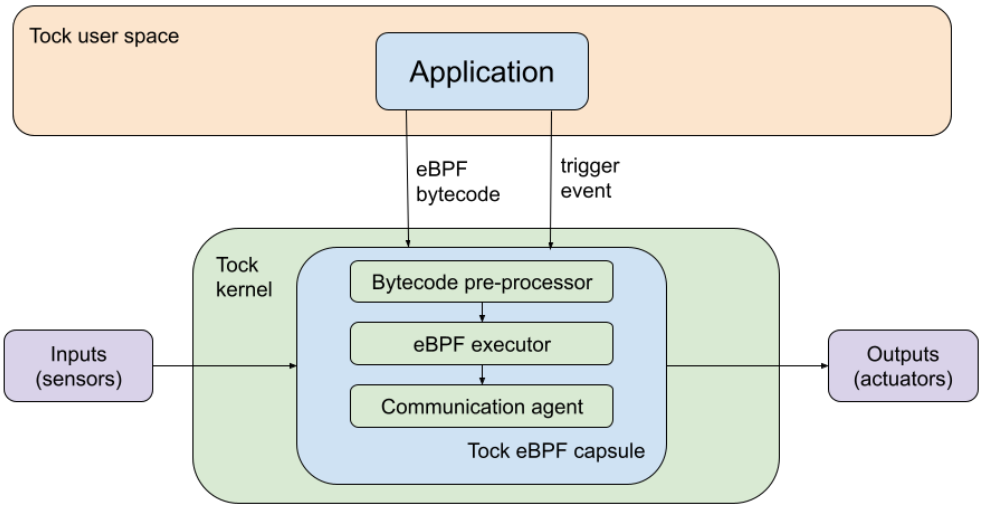

Figura 8: Arhitectura sistemului propus.

Implementarea propusă se bazează pe următoarele trei componente principale:

- Executorul eBPF - Aceasta este capsula care execută rutina de gestionare a întreruperilor și care are nevoie de interacțiunea cu kernel-ul Tock și cu spațiul utilizator.
- Preprocesorul de bytecode - Modifică bytecode-ul eBPF original pentru a fi compatibil cu modelul de memorie Tock.
- Agentul de comunicare a spațiului utilizator cu sandbox-ul eBPF - Acesta este API-ul capsulei prin care bytecodul definit este injectat în executor.

5.3.1 Executorul eBPF

Componenta centrală în arhitectura pe care o propunem este executorul de bytecode eBPF dezvoltat ca o capsulă Tock.

În acest context, am identificat următoarele constrângeri privind implementarea executorului, care se datorează regulilor de implementare ale Tock:

- C1 - Bytecode-ul injectat trebuie să aibă un timp de execuție determinist.
- C2 - La acest nivel nu este permis codul Rust nesigur.
- C3 - În Tock nu este permisă alocarea de memorie pe heap, executorul eBPF nu poate utiliza heap-ul.

Pentru implementare, am folosit rbpf [51], un proiect open-source care își propune să furnizeze un executor eBPF scris în întregime în Rust.

Pentru abordarea noastră, sandbox-ul eBPF a trebuit să fie rulat în kernel ca o capsulă. Acest lucru ne-a condus la evidențierea a trei dezavantaje principale în implementarea originală rbpf, care nu sunt potrivite pentru cazul de utilizare pe care îl propunem.

- Rbpf depinde de biblioteca standard Rust și se bazează pe structura Vec, care utilizează heap pentru a aloca date. Acest lucru contrazice C3.
- Rbpf are blocuri mari de cod nesigur, ceea ce contrazice C2 din lista de constrângeri.
- Proiectul complet rbpf are dimensiuni foarte mari. În cazul nostru, memoria disponibilă este prea mică pentru a găzdui toate caracteristicile. În plus, multe dintre caracteristicile "nice-to-have" incluse în rbpf nu sunt necesare pentru cazul nostru de utilizare, cum ar fi JIT sau funcțiile de ajutor.

În acest context, am generat o versiune personalizată de rbpf care are caracteristicile necesare pentru a fi integrată în siguranță în kernel Tock. Pentru a obține versiunea personalizată, am urmat anumiți pași specifici de implementare:

1. Am șters toate caracteristicile inutile nouă, memoria eBPF este reprezentată ca un șir în programul de bytecode.
2. Am rescris toate părțile de cod care depind de biblioteca standard Rust, am înlocuit reprezentarea memoriei cu un șir mutabil de numere întregi fără semn pe 8 biți.
3. Am eliminat toate dereferențările de pointeri brute, care produc cod nesigur care nu poate fi rulat în interiorul capsulelor Tock.

După modificările menționate mai sus, versiunea rbpf personalizată are toate capacitățile necesare pentru a fi rulată în siguranță ca o capsulă Tock deasupra dispozitivelor încorporate care au capacități constrânse.

5.3.2 Preprocesarea bytecode-ului

În plus față de sandbox-ul eBPF, un aspect important în rularea codului sigur în kernel este modul în care este generat bytecodul eBPF care urmează să fie executat.

ISA-ul eBPF este simplu, programul binar în sine fiind o secvență lungă de instrucțiuni pe 64 de biți care trebuie să respecte formatul prezentat în figura 9.

Preprocesorul de bytecode face parte din sandbox-ul rbpf pe care l-am folosit pentru implementarea noastră.

Modificarea majoră este legată de alocarea bufferului de memorie. Inițial, executorul reprezenta zona de memorie transmisă din spațiul utilizator ca o structură *Vec*, care este alocată pe heap în timpul execuției. Acest lucru contrazice C3. Prin urmare, am modificat implementarea inițială a gestionării memoriei pentru a înlocui structura *Vec* cu un *array*, care este alocat pe stivă.

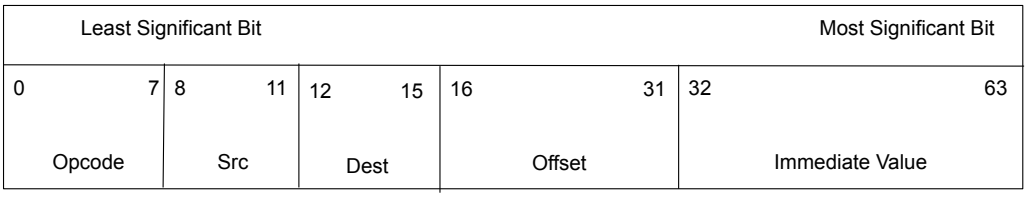

Figura 9: Formatul instrucțiunilor eBPF.

Implementarea inițială a rbpf utilizează o structură *Vec* pentru stiva mașinii virtuale. Soluția noastră a fost să alocăm un *array* în capsula principală și să îl transmitem executorului împreună cu structura de memorie. Cele două au fost concatenate și transmise mai departe. Prin urmare, una dintre provocări este de a ne asigura că operațiile legate de gestionarea memoriei sunt efectuate corect.

5.3.3 Agentul de comunicare din spațiul utilizator cu executorul eBPF

Ultima componentă a arhitecturii propuse este legată de integrarea efectivă a executorului eBPF în stiva Tock. Executorul a fost implementat sub forma unei capsule care expune un API pentru spațiul utilizator (Figura 10), ceea ce face ca integrarea să fie mai complexă.

Capsula pe care am creat-o este menită să ofere o implementare generică ce permite interacțiuni cu toate celelalte capsule Tock existente. În acest context, această capsulă nu controlează direct hardware-ul, ci trimite comenzi către capsulele de control hardware deja implementate. Cu toate acestea, ea trebuie să asocieze codul de octet cu operațiile hardware necesare, să identifice capsula de control hardware corespunzătoare și să definească comenzile necesare. În continuare, citește rezultatul pe care îl returnează capsula de control hardware și îl transmite înapoi în spațiul utilizator.

Pentru comunicarea cu spațiul de utilizator, capsula primește bytecode-ul de la aplicație prin intermediul unui apel de sistem. Bytecode-ul propriu-zis este transmis sub forma unui șir de octeți stocat în interiorul unui buffer partajat de kernel și de aplicație.

5.4 Teste și rezultate

Pentru a evalua eficiența abordării propuse, am măsurat latența sistemului la gestionarea întreruperilor.

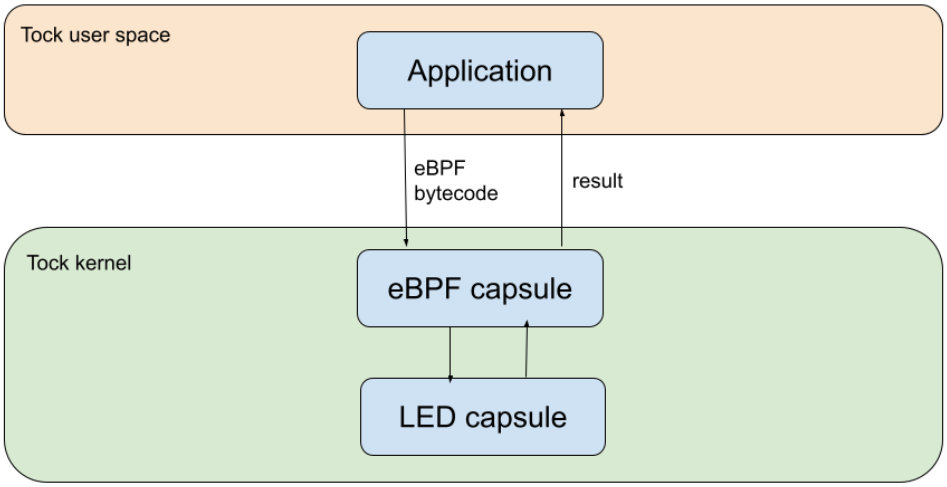

Figura 10: Comunicarea dintre aplicație și o capsulă periferică folosind eBPF.

.

5.4.1 Evaluarea implementării originale

Am efectuat mai întâi teste pentru a evalua performanța sistemului de operare Tock pe diferite arhitecturi. Aceste teste au fost menite să identifice latențele originale în kernel-ul Tock și comportamentul general al sistemului în gestionarea intreruperilor externe. Pentru a realiza acest lucru, am implementat două categorii de teste:

1. Comportament general - Acestea sunt teste de stres menite să identifice dacă un dispozitiv care rulează Tock poate gestiona o cantitate masivă de intreruperi externe la o frecvență înaltă.
2. Măsurarea latenței - Aceste teste s-au concentrat pe evaluarea latenței în gestionarea întreruperilor în Tock.

Pentru toate testele, am folosit un dispozitiv BBC micro:bit v2, care are un MCU nRF52833.

Teste de comportament general

. Pentru aceste teste, am generat valori alternative HIGH-LOW la diferite frecvențe, în timp ce dispozitivul micro:bit rulează unul sau mai multe procese care gestionează întreruperile primite. Rutina de gestionare a întreruperilor incrementează o valoare și o imprimă în consola serială.

Pentru cazul în care dispozitivul micro:bit rulează o singură aplicație a cărei singur scop este să gestioneze aceste întreruperi primite, am reușit să gestionăm toate intreruperile primite la o frecvență de 2 KHz. Pentru frecvențe mai mari, se pierd aproximativ 30% din mesajele adișate. În cele din urmă, la o frecvență de 10 KHz, sistemul nu afișează niciun mesaj, deoarece este

prea ocupat pentru a gestiona întreruperile venite de la osciloscop, iar funcția de afișare nu este niciodată apelată.

Am făcut același test cu un sistem care rulează simultan două aplicații paralele. Am rulat aplicația menționată anterior care gestionează întreruperi în paralel cu o aplicație care face ca un LED să clipească o dată pe secundă. Frecvențele de întrerupere la care funcționează sistemul sunt aceleași, în timp ce pentru frecvențe mai mari de 2 KHz, se pierd până la 70% din întreruperi. La fel ca în primul caz, sistemul nu mai afișează mesaje pentru întreruperile generate la o frecvență de 10 KHz. Cu toate acestea, procesul de clipire a LED-ului funcționează în continuare.

Testul final înlocuiește aplicația de clipire a LED-ului cu una care înregistrează o rutină de întrerupere pentru un buton. În acest caz, pentru întreruperi generate la o frecvență de 2 KHz, comportamentul general arată că ambele rutine, cea pentru pinul conectat la osciloscop și cea pentru pinul conectat la buton, sunt apelate. Mai trebuie să investigăm cauza care duce la această apariție.

Aceste teste concluzionează că Tock nu este conceput pentru a gestiona întreruperi de înaltă frecvență și că mecanismul de gestionare a întreruperilor nu este optimizat pentru răspunsuri rapide.

Măsurători de latență

Pentru a evalua latențele de gestionare a întreruperilor specifice lui Tock, am cronometrat timpul de răspuns între apelurile de sistem (syscall).

Toate măsurătorile au fost calculate ca medie a 150 de eșantioane diferite. Abaterea măsurătorilor a fost de aproximativ 150 µs.

La nivelul spațiului utilizator, am implementat patru scenarii:

1. Un singur proces din spațiul utilizator - Dispozitivul rulează un singur proces care emite în mod continuu o comandă syscall la fiecare 250 de milisecunde.
2. Trei procese identice din spațiul utilizator - Dispozitivul rulează trei procese diferite, fiecare dintre ele lansând o comandă syscall la fiecare 250 de milisecunde.
3. Un proces cu utilizare intensivă a CPU - Pentru a pune mai multă presiune asupra sistemului, dispozitivul rulează un proces cu utilizare intensivă a procesorului (o buclă fără întârzieri) și două procese care emit o comandă syscall la fiecare 250 de milisecunde.
4. Două procese cu utilizare intensivă - Acest test de stres utilizează două procese cu utilizare intensivă a procesorului și un proces blocant, similar celor prezentate mai sus.

Pentru fiecare scenariu, am efectuat două măsurători diferite: una pentru un syscall sincron și una pentru un syscall asincron.

Măsurarea sincronă se concentrează asupra duratei de transmitere a unui syscall din spațiul utilizator către kernel și de primire a rezultatului de către spațiul utilizator.

Măsurarea asincronă urmărește durata în care un syscall ajunge la kernel și se întoarce în spațiul utilizator, dar în cazul unei capsule asincrone.

Rezultatele măsurătorilor sunt afișate în tabelul 5. Întârzierile obținute sunt considerabil mai mari decât întârzierile specifice unui sistem în timp real cu latență redusă, unde valorile sunt în jur de 50 de microsecunde [52].

Tabel 5: Măsurători de latență pentru un dispozitiv micro:bit.

	Un process	**Trei procese**	**Un proces intensiv**	**Două procese intensive**
Măsurătoare sinconă	5127 µs	90127 µs	91452 µs	125250 µs
Măsurătoare asincornă	9213 µs	91545 µs	90643 µs	120903 µs

5.4.2 Evaluarea abordării propuse

Evaluarea sistemului implementat a fost realizată în mod incremental. Componentele specifice ale sistemului au fost evaluate comparativ, precum și soluția globală.

Teste de eficiență a executorului eBPF

Primele teste se concentrează pe compararea rezultatelor obținute la rularea bytecode-ului eBPF utilizând implementarea originală rbpf cu cea adaptată de noi. Acest lucru asigură validitatea soluției propuse.

În acest scop, am creat mai multe aplicații C pentru fiecare operațiune de încărcare și stocare, le-am compilat în eBPF, apoi am rulat bytecode-ul într-o aplicație Rust [53]. Acest lucru ne-a permis să comparăm rezultatul rbpf cu rezultatul obținut din implementarea noastră modificată de rbpf. Aceste teste au fost rulate pe un sistem de uz general, capabil să ruleze ambele versiuni ale implementării rbpf.

Am utilizat același cadru pentru a măsura viteza de execuție a instrucțiunilor de încărcare și stocare în executorul rbpf original în comparație cu cel pe care îl propunem.

Tabelul 6 prezintă rezultatele, în care timpii obținuți de executorul eBPF pe care l-am implementat sunt, în medie, de 3-4 ori mai mici pentru operațiile simple și de aproximativ 2-5 ori mai mici pentru cele mai complexe.

Tabel 6: Comparația vitezei de execuție a operațiunilor de încărcare și stocare între executorul rbpf original și cel modificat.

Nume test	Descriere	Executorul Rbpf original	Executorul Rbpf modificat
LD_ST_DW_REG	Load and Store Double-Word into Reg	2701 µs	660 µs
LD_ABS_DW	Load Double-Word from absolute indexed address	1415 µs	490 µs
ST_DW_IMM	Store Double-Word to absolute indexed address	1986 µs	500 µs
LD_IND_DW	Load Double-Word from indirect indexed address	1698 µs	293 µs
Stack test	Generate a vector of 496 char elements on the stack with values from 0 to 495	75,159 µs	28,358 µs

Testarea platformei

Pentru evaluarea abordării complete propuse, am folosit același dispozitiv micro:bit v2 ca și în testele inițiale. Am definit mai multe seturi de teste pentru a evalua atât reacția sistemului la gestionarea rapidă a întreruperilor, cât și la întârzierea în tratarea întreruperilor. Pentru fiecare test, am efectuat 50 de măsurători și am calculat valoarea medie, care este prezentată mai jos. Abaterea în rezultatele obținute a fost de 5 µs.

Am testat capacitatea de reacție a sistemului la primirea de întreruperi la o frecvență de 10 KHz. În timpul acestor teste curente, toate întreruperile venite la o frecvență de 10 KHz au fost gestionate cu succes.

În continuare, ne-am concentrat asupra a două categorii principale de teste pentru a evalua timpul de răspuns. Aceste teste au fost efectuate pentru a compara timpul de răspuns al abordării bazate pe eBPF cu timpul de răspuns al abordării Tock standard și cu o capsulă concepută pentru a gestiona aceleași întreruperi. Rezultatele sunt prezentate în tabelul 7.

Tabel 7: Comparație între timpii de latență obținuți.

	Tock Standard	Capsulă	Executorul eBPF
Un pin GPIO	104 µs	14 µs	60 µs
Șir de pini	136 µs	43 µs	208 µs

Figura 11 prezintă, de asemenea, o reprezentare vizuală a rezultatelor.

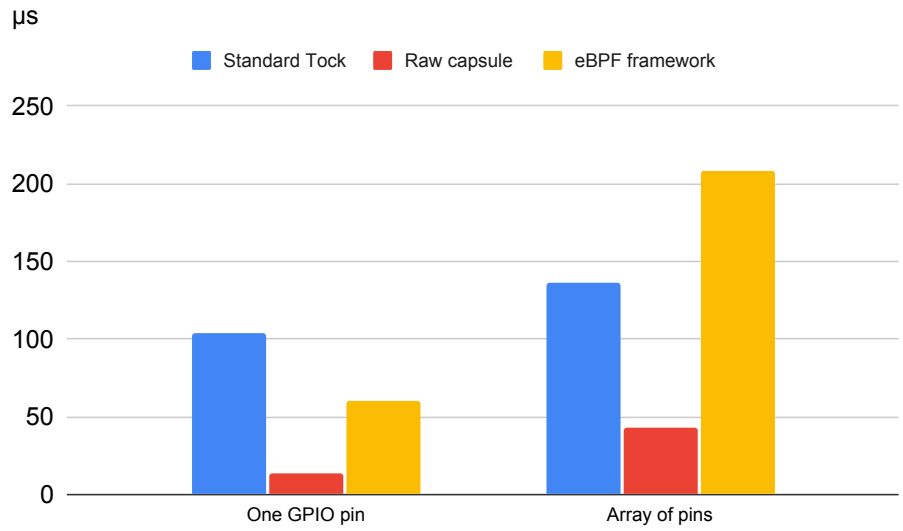

Figura 11: Întârzierile de gestionare a întreruperilor.

Rezultatele obținute prin abordarea pe care o propunem au fost comparate cu întârzierile măsurate în implementarea originală Tock și cu cele rezultate în cazul unei capsule dedicate pentru gestionarea acestor întreruperi. Așa cum era de așteptat, întârzierile în cazul unei capsule dedicate sunt cele mai mici, deoarece toată logica aplicației este implementată în kernel.

În comparație cu implementarea Tock originală, abordarea eBPF este mai rapidă pentru primul caz de utilizare, adică atunci când pinii GPIO sunt predefiniți. Cu toate acestea, în cel de-al doilea caz de utilizare, atunci când se utilizează un șir de pini, acest timp de răspuns crește semnificativ. Acest lucru se datorează operațiunii de iterare a șirului. În Rust, iterarea unui șir consumă foarte mult timp. Prin urmare, principala îmbunătățire viitoare asupra căreia ne vom concentra această cercetare este reducerea acestei cost suplimentare.

5.4.3 Rezultate

Rezultatele finale evidențiază o îmbunătățire semnificativă a răspunsului sistemului la întreruperile cu o frecvență înaltă. Mai precis, am implementat o abordare care permite kernelui Tock să ruleze cod personalizat declanșat de întreruperi cu o frecvență de 10 KHz, în timp ce kernel original se blochează în timpul unui astfel de caz de utilizare.

În comparație cu abordarea brută de introducere a unei capsule personalizate în kernel destinată să gestioneze anumite întreruperi, soluția bazată pe eBPF are un timp de răspuns mai mare. Cu toate acestea, principalul avantaj al soluției propuse este generalitatea sa. Ea nu implică o capsulă diferită pentru fiecare caz de utilizare diferită, ci permite ca rutina de întrerupere să fie injectată în kernel din spațiul utilizator.

În ceea ce privește întârzierea în tratarea unui eveniment de intrare, abordarea pe care o propunem măsoară o valoare medie de 200 µs între declanșarea întreruperii și schimbarea stării unui pin. Această valoare se referă la cazul în care lucrăm cu un șir de pini care sunt citiți. Dacă revenim la un caz de utilizare specific în care pinii sunt definiți static în capsulă, întârzierile scad la 60 µs, ceea ce este comparabil cu alte sisteme în timp real. Evaluarea realizată de Zhang M. et al. [52] conturează faptul că un sistem în timp real implementat folosind un Raspberry Pi 3 care are un CPU de 1,2 GHz și un BeagleBone Black cu un CPU de 1 GHz are o latență de răspuns între 45 și 75 µs.

6 CONCLUZII

Această teză se concentrează în jurul mijloacelor de securizare a infrastructurilor Internet of Things prin luarea în considerare a diversității tehnologiilor implicate în construirea unui astfel de sistem. Prin urmare, lucrarea noastră abordează mai multe aspecte legate de IoT, pornind de la sisteme de operare sigure și urcând în stiva IoT până la tehnologiile cloud implicate în întreținerea oricărui dispozitiv Internet of Things.

În prima noastră contribuție, ne concentrăm pe rularea limbajelor de programare moderne și sigure pentru anumite calculatoare și microcontrollere integrate. De exemplu, am reușit să rulăm JerryScript pe NXP IoT Rapid Prototyping Kit, un dispozitiv conceput pentru prototiparea aplicațiilor IoT pentru case inteligente și stații meteo. Am cercetat, de asemenea, rularea unor astfel de limbaje de programare de nivel înalt pe dispozitive cu mai multe constrângeri și pe sisteme de operare reduse, cu scopul de a aborda aspectul securității atât la nivelul kernelului, cât și la nivelul aplicațiilor. În acest scop, am reușit să rulăm aplicații scrise în D-lang pe Tock, un sistem de operare securizat pentru microcontrollere scris în Rust.

Un alt aspect complementar pe care l-am abordat este reprezentat de limbajele de programare vizuală. Ne-am concentrat pe o soluție de programare vizuală frecvent utilizată pentru IoT: Node-RED. Aceasta are o abordare bazată pe fluxuri, programatorii folosind noduri conectate pentru a defini comportamentul infrastructurilor lor și modul în care acestea interacționează între ele. În această lucrare de cercetare, am propus o alternativă la Node-RED ca soluție bazată pe BPMN pentru definirea sistemelor IoT. Datorită constrângerilor sale specifice și a utilizării unui interpretOR, platforma bazată pe BPMN pe care o propunem este sigură din punctul de vedere al aplicațiilor.

Deoarece securitatea este un aspect care trebuie luat în considerare încă din faza de prototipare în ciclul de dezvoltare a unui produs, ne concentrăm următoarea contribuție pe identificarea și propunerea de tehnologii care să permită integratorilor realizarea de prototipuri IoT sigure și robuste. Ca urmare, propunem Wyliodrin, o platformă de prototipare pe care am proiectat-o pentru a construi și implementa în siguranță și în mod eficient aplicații pe calculatoare integrate, cum ar fi Raspberry Pi, BeagleBone Black sau Qualcomm DragonBoard.

Am dezvoltat Wyliodrin ca o platformă generică și ușor extensibilă și am adaptat-o pentru a funcționa cu diverse tehnologii hardware și software. Prin urmare, această contribuție răspunde la întrebarea referitoare la modalitățile de aplicare a securității sistemelor IoT în timpul ciclului de viață al produsului.

Extinzând instrumentele de prototipare IoT propuse, am conceput o versiune Wyliodrin specifică pentru educație. Deși sunt similare în multe privințe, platformele de prototipare și cele

educaționale au, de asemenea, anumite caracteristici specifice pentru fiecare dintre scopurile lor.

Soluția finală constă într-o platformă hardware și software open-source modulară care poate fi ușor utilizată de către comunitate și de către educatori. Ca urmare, în intervalul 1 octombrie 2019 - 31 decembrie 2022, un total de 30.000 de utilizatori au dezvoltat aplicații folosind-o.

O altă contribuție abordează vulnerabilitățile de securitate legate de implementarea și actualizările de software în dispozitivele Internet of Things. Această contribuție este împărțită în două secțiuni principale. În prima secțiune, propunem o infrastructură generică de actualizări care se bazează pe un model matematic. A doua secțiune detaliază implementarea acestui model folosind tehnologii de ultimă generație, cum ar fi docker. Pentru modelul matematic, luăm în considerare toate aspectele legate de implementarea de software în IoT, cu accent pe securitate. De asemenea, luăm în considerare fiabilitatea unei astfel de infrastructuri și ne concentrăm asupra modelelor care previn blocarea unui dispozitiv în mijlocul unei actualizări.

În continuare, pentru a demonstra fezabilitatea modelului general pe care îl propunem, construim o implementare a unei platforme generice, open-source, de instalare și actualizare, pe care am folosit-o pentru a efectua 13.300 de actualizări de software pe dispozitive de pe trei continente diferite, cu o rată de succes de peste 70% și fără dispozitive blocate.

Ultima noastră lucrare prezentată în această teză abordează nivelul cel mai de jos din stiva IoT, concentrându-se pe dispozitivele cu capacități reduse, cum ar fi microcontrollerele, și pe securitatea sistemelor de operare care rulează pe acestea.

Valorificăm avantajele lui Tock, care este un sistem de operare open-source pentru microcontrollere cu un puternic fundament de securitate, dar căruia îi lipsește caracteristica de timp real. Astfel, am cercetat modalități de introducere a unui modul de operații in timp real, astfel încât să putem obține un sistem de operare care să fie atât sigur, cât să suporte operații în timp real. Abordarea pe care am adoptat-o a fost aceea de a valorifica tehnologia eBPF utilizată pentru procesarea traficului de rețea și de a o folosi pentru gestionarea operațiilor cu latență redusă. Rezultatele pe care le-am obținut sunt comparabile cu FreeRTOS, cel mai utilizat sistem de operare în timp real pentru microcontrollere.

BIBLIOGRAFIE

[1] J. Margolis, T. T. Oh, S. Jadhav, Y. H. Kim, and J. N. Kim, "An In-Depth Analysis of the Mirai Botnet," in *2017 International Conference on Software Security and Assurance (ICSSA)*, pp. 6–12, 2017.

[2] M. Antonakakis, T. April, M. Bailey, M. Bernhard, E. Bursztein, J. Cochran, Z. Durumeric, J. A. Halderman, L. Invernizzi, M. Kallitsis, *et al.*, "Understanding the mirai botnet," in *26th USENIX security symposium (USENIX Security 17)*, pp. 1093–1110, 2017.

[3] G. Thomas, "A proactive approach to more secure code." Available online: https://msrc-blog.microsoft.com/2019/07/16/a-proactive-approach-to-more-secure-code (accessed on 2 December 2022).

[4] A. Taylor, A. Whalley, D. Jansens, and N. Oskov, "An update on Memory Safety in Chrome." Available online: https://security.googleblog.com/2021/09/an-update-on-memory-safety-in-chrome.html (accessed on 2 December 2022).

[5] R. J. Walls, N. F. Brown, T. Le Baron, C. A. Shue, H. Okhravi, and B. C. Ward, "Control-flow integrity for real-time embedded systems," in *31st Euromicro Conference on Real-Time Systems (ECRTS 2019)*, Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik, 2019.

[6] A. Taivalsaari and T. Mikkonen, "A roadmap to the programmable world: software challenges in the IoT era," *IEEE software*, vol. 34, no. 1, pp. 72–80, 2017.

[7] R. Avila, "Embedded Software Programming Languages: Pros, Cons, and Comparisons of Popular Languages." Available online: https://www.qt.io/embedded-development-talk/embedded-software-programming-languages-pros-cons-and-comparisons-of-popular-languages (accessed on 2 December 2022).

[8] S. Bhartiya, "Linux in 2020: 27.8 million lines of code in the kernel, 1.3 million in systemd." Available online: https://www.linux.com/news/linux-in-2020-27-8-million-lines-of-code-in-the-kernel-1-3-million-in-systemd/ (accessed on 2 December 2022).

[9] "Mozilla Welcomes the Rust Foundation." Available online: https://blog.mozilla.org/en/mozilla/mozilla-welcomes-the-rust-foundation (accessed on 2 December 2022).

[10] "Tock Embedded Operating System." Available online: https://www.tockos.org (accessed on 2 December 2022).

[11] "Hubris." Available online: https://hubris.oxide.computer (accessed on 2 December 2022).

[12] "Redox." Available online: https://www.redox-os.org (accessed on 2 December 2022).

[13] "FreeRTOS." Available online: https://www.freertos.org (accessed on 2 December 2022).

[14] "Zephyr Project." Available online: https://www.zephyrproject.org (accessed on 2 December 2022).

[15] T. Severin, I. Culic, and A. Radovici, "Enabling High-Level Programming Languages on IoT Devices," in *2020 19th RoEduNet Conference: Networking in Education and Research (RoEduNet)*, pp. 1–6, IEEE, 2020.

[16] "Jerryscript engine for internet of things." Available online: https://jerryscript.net (accessed on 2 December 2022).

[17] I. Culic, A. Radovici, L. Moraru, C. Radu, and J.-A. Vaduva, "Porting JerryScript to NXP Rapid Prototyping Kit," *eLearning & Software for Education*, vol. 2, 2020.

[18] I. Culic and A. Radovici, "Development platform for building advanced Internet of Things systems," in *2017 16th RoEduNet Conference: Networking in Education and Research (RoEduNet)*, pp. 1–5, 2017.

[19] A. Radovici and I. Culic, "Open cloud platform for programming embedded systems," in *2013 RoEduNet International Conference 12th Edition: Networking in Education and Research*, pp. 1–5, IEEE, 2013.

[20] I. Culic, A. Radovici, and C. Dumitru, "Hardware Simulator for Teaching Internet of Things," *eLearning & Software for Education*, vol. 2, 2020.

[21] I. Culic and A. Radovici, "Open Source Technologies in Teaching Internet of Things," *eLearning & Software for Education*, vol. 2, 2017.

[22] A. Radovici, I. Culic, D. Rosner, and F. Oprea, "A model for the remote deployment, update, and safe recovery for commercial sensor-based IoT systems," *Sensors*, vol. 20, no. 16, p. 4393, 2020.

[23] A. Vochescu, I. Culic, and A. Radovici, "Multi-Layer Security Framework for IoT Devices," in *2020 19th RoEduNet Conference: Networking in Education and Research (RoEduNet)*, pp. 1–5, IEEE, 2020.

[24] I. Culic, A. Vochescu, and A. Radovici, "A Low-Latency Optimization of a Rust-Based Secure Operating System for Embedded Devices," *Sensors*, vol. 22, no. 22, p. 8700, 2022.

[25] "D Programming Language." Available online: https://dlang.org (accessed on 3 December 2022).

[26] E. Staniloiu, A. Militaru, R. Nitu, and R. Deaconescu, "Safer Linux Kernel Modules using the D Programming Language," *IEEE Access*, pp. 1–1, 2022.

[27] "Node-RED." Available online: https://nodered.org/ (accessed on 4 December 2022).

[28] "bpmn-js source code." Available online: https://github.com/bpmn-io/bpmn-js (accessed on 26 December 2022).

[29] "Blockly." Available online: https://developers.google.com/blockly (accessed on 4 December 2022).

[30] "Eclipse/Mraa - Github." Available online: https://github.com/eclipse/mraa (accessed on 5 January 2023).

[31] "jison - Github." Available online: https://github.com/zaach/jison (accessed on 5 January 2023).

[32] D. Grune, K. Van Reeuwijk, H. E. Bal, C. J. Jacobs, and K. Langendoen, *Modern compiler design*. Springer Science & Business Media, 2012.

[33] "Wyliodrin - Github." Available online: https://github.com/wyliodrinstudio/WyliodrinSTUDIO (accessed on 7 January 2023).

[34] "Wyliodrin." Available online: https://wyliodrin.studio/ (accessed on 7 January 2023).

[35] "Buy a Raspberry Pi - Raspberry Pi." https://www.raspberrypi.org/products (accessed on 19 February 2023).

[36] E. Crawley, J. Malmqvist, S. Ostlund, D. Brodeur, and K. Edstrom, "Rethinking engineering education," *The CDIO approach*, vol. 302, no. 2, pp. 60–62, 2007.

[37] G. T. Heydt and V. Vittal, "Feeding our profession [power engineering education]," *IEEE Power and Energy Magazine*, vol. 1, no. 1, pp. 38–45, 2003.

[38] A. Radovici, I. Culic, O. Stoica, and D. Rosner, "Building a Smart City Infrastructure using Raspberry Pi and Arduino," 2016.

[39] B. Burd, L. Barker, F. A. F. Pérez, I. Russell, B. Siever, L. Tudor, M. McCarthy, and I. Pollock, "The internet of things in undergraduate computer and information science education: exploring curricula and pedagogy," in *Proceedings Companion of the 23rd Annual ACM Conference on Innovation and Technology in Computer Science Education*, pp. 200–216, 2018.

[40] J. Sobota, R. PiŜl, P. Balda, and M. Schlegel, "Raspberry Pi and Arduino boards in control education," *IFAC Proceedings Volumes*, vol. 46, no. 17, pp. 7–12, 2013.

[41] S. L. Kinney, *Trusted platform module basics: using TPM in embedded systems*. Elsevier, 2006.

[42] S. Pinto and N. Santos, "Demystifying Arm TrustZone: A Comprehensive Survey," *ACM Comput. Surv.*, vol. 51, 01 2019.

[43] V. Costan and S. Devadas, "Intel SGX Explained.," *IACR Cryptology ePrint Archive*, vol. 2016, no. 086, pp. 1–118, 2016.

[44] N. Koblitz, "Elliptic curve cryptosystems," *Mathematics of computation*, vol. 48, no. 177, pp. 203–209, 1987.

[45] R. L. Rivest, A. Shamir, and L. Adleman, "A method for obtaining digital signatures and public-key cryptosystems," *Communications of the ACM*, vol. 21, no. 2, pp. 120–126, 1978.

[46] "IoTWay." Available online: https://iotway.io (accessed on 19 February 2023).

[47] P. Bourhis, J. L. Reutter, F. Suárez, and D. Vrgoč, "JSON: data model, query languages and schema specification," in *Proceedings of the 36th ACM SIGMOD-SIGACT-SIGAI symposium on principles of database systems*, pp. 123–135, 2017.

[48] B. Leiba, "OAuth Web Authorization Protocol," *IEEE Internet Computing*, vol. 16, no. 1, pp. 74–77, 2012.

[49] "Tock source code - Github." Available online: https://github.com/tock/tock (accessed on 22 September 2022).

[50] S. Miano, M. Bertrone, F. Risso, M. Tumolo, and M. V. Bernal, "Creating complex network services with ebpf: Experience and lessons learned," in *2018 IEEE 19th International Conference on High Performance Switching and Routing (HPSR)*, pp. 1–8, 2018.

[51] "rbpf - Github." Available online: https://github.com/qmonnet/rbpf (accessed on 22 September 2022).

[52] M. Zhang, M. Timmerman, L. Perneel, and T. Goedemé, "Which is the best real-time operating system for drones? evaluation of the real-time characteristics of nuttx and chibios," in *2021 International Conference on Unmanned Aircraft Systems (ICUAS)*, pp. 582–590, 2021.

[53] "rbpf_tests - Github." Available online: https://github.com/WyliodrinEmbeddedIoT/rbpf_tests (accessed on 22 September 2022).