eman ta zabal zazu

Universidad del País Vasco
Euskal Herriko Unibertsitatea

Informatika Ingeniaritzako Gradua

Konputagailuen Ingeniaritza

Gradu Amaierako Proiektua

CoAP liburutegien konparaketa praktiko eta teorikoa

Egilea: *Markel Iglesias*

Tutorea: *Amaya Ibarra*

Enpresako tutorea: *Aitor Urbieta*

informatika fakultatea
facultad de informática

2017

Abstract

Over the last years, the Internet of Things (IoT) has grown in protocols, implementation and use cases. Regarding to communication protocols, the Constrained Application Protocol (CoAP) has started to gain some momentum as it follows the REST paradigm, it is very lightweight and it can be embedded in constrained device and networks. There are several available open source libraries that implement CoAP, that target different platforms and written in different programming languages. Each of these libraries has its own features and requirements and therefore, it is very important to select the most appropiate one for each proyect or system. The aim of the work carried out in this proyect has been to help developers choose the library that suits better each system. To do that, a preliminary research has been done on libraries, they have been theoretically analyzed and finally some "demos"have been developed. After analyzing the features and extensions each library supports, the "demos"have been executed to measure some important parameters such as latency and memory and CPU consumption, in a concrete hardware. With this analysis and measurements, the goal of helping developers select the correct library for their system has been targeted. Finally, a paper has been written and submitted to an international conference.

Resumen

Los últimos años, el Internet de las Cosas o Internet of Things (IoT) ha crecido en protocolos, implementaciones y casos de uso. Entre los protocolos de comunicación, Constrained Application Protocol (CoAP) ha empezado a ganar fuerza ya que sigue el paradigma REST, es muy ligero y se puede incluir en dispositivos y redes que tengan ciertas limitaciones. Hay varias librerías de código abierto que implementan CoAP para diferentes plataformas y lenguajes de programación. Cada librería tiene sus características y requisitos y es muy importante elegir la implementación adecuada para cada aplicación o sistema. El trabajo realizado en este proyecto se ha enfocado a cumplir ese objetivo, facilitar a los desarrolladores la elección de librerías para cada sistema particular. Para ello, se ha hecho un estudio previo de las librerías existentes, se han analizado teóricamente y finalmente se han desarrollado unas "demos". Después de analizar las características y extensiones que soporta cada librería, se han ejecutado las "demos"para tomar medidas de parámetros importantes como la latencia y el consumo de memoria y CPU, en un hardware concreto. Con este análisis y con estas medidas, el objetivo de facilitar la elección de la mejor librería para cada aplicación se ha cumplido. Además, se ha escrito un artículo científico que se ha enviado para su publicación en una conferencia internacional.

Laburpena

Azken urteotan, Gauzen Internet edo Internet of Things (IoT) zabaltzen joan da eta hainbat protokolo, inplementazio eta erabilpen kasu garatu dira. Komunikazio protokoloen artean, Contrained Application Protocol (CoAP) nabarmentzen hasi da, REST paradigma jarraitzen duelako, oso arina delako eta gailu eta sare oso mugatuetan sar daitekeelako. CoAP inplementatzen duten hainbat liburutegi libre daude eskuragarri, plataforma eta programazio lengoaia ezberdinetan. Horietako liburutegi bakoitzak bere ezaugarri eta eskakizunak dauzka eta oso garrantzitsua da aplikazio edo sistema bakoitzean liburutegi egokia aukeratzea. Proiektu honetan aurrera atera den lana zentzu honetan joan da. Hori dela eta, liburutegi ezberdinak bilatu, teorikoki aztertu eta bukatzeko "demo"batzuk prestatu dira, neurri ezberdinak hartzeko. Horrela, liburutegi bakoitzak eskaintzen dituen ezaugarriak eta gehigarriak aztertzen dira lehenengo eta gero, demoak martxan jarri eta latentzia eta memoria eta CPU erabilera neurtu dira, hardware konkretu batean. Era honetan, garatzaileei beraien sisteman inplementatzeko liburutegi egokiena aukeratzen laguntzea izanik lanaren helburu nagusia, artikulu zientifiko bat idatzi eta nazioarteko konferentzia batera bidali da.

Akronimoak

ACK	Acknowledgement
AMQP	Advanced Message Queuing Protocol
CoAP	Constrained Application Protocol
CoRE	Constrained RESTful Environments
CPU	Central Processing Unit
DDS	Data Distribution Service
DTLS	Datagram Transport Layer Security
HTTP	Hypertext Transfer Protocol
IANA	Internet Assigned Numbers Authority
IETF	The Internet Engineering Task Force
IoT	Internet of Things
IP	Internet Protocol
JMS	Java Message Service
MQTT	MQ Telemetry Transport
MQTT-SN	MQTT For Sensor Networks
QoS	Quality of Service
RAM	Random Access Memory
RD	Resource Directory
REST	Representational State Transfer
ROM	Read Only Memory
TCP	Transmission Control Protocol
TLS	Transport Layer Security
UDP	User Datagram Protocol
URI	Uniform Resource Identifier
XMPP	Extensible Messaging and Presence Protocol

Gaien aurkibidea

Irudien aurkibidea

Taulen aurkibidea

1. KAPITULUA

Sarrera

Dokumentu hau Informatika Ingeneritzako Graduko bukaerako proiektuaren memoria da, IK4-Ikerlan Ikerketa Teknologikoen Zentruan burutu dena.

1.1 CoAP protokoloa

Gauzen Internet edo Internet of Things (IoT), geroz eta garatuago dago eta gertuago dago gizartean garrantzi handia edukiko duen momentua. Eguneroko objektuak konektatzen dituen sarea da IoT, bai beraien artean eta baita ohizko internet-era ere bai. Eguneroko objektuan esaterakoan, ez dira gailu elektronikoak bakarrik (telefono mugikorrak, pultsera azkarrak...) edo barnean elektronika konplexua daukatenak (autoak edo etxetresna handiak), baizik eta beste edozein produktu, esate baterako janaria, arropa, materialak edo piezak.

Gauzen sare hori sortzeko, gaur egun ohizkoak diren interneteko protokoloak ez dira egokiak, energia eta konputazio kontsumo handia baitaukate, beraz hainbat protokolo berri agertu dira azken urteotan, arazo horri aurre egiteko. Horietako protokolo bat da Constrained Application Protocol (CoAP), garatzaileek HTTP-n ikasitakoan oinarrituta eta antzeko modeloa jarraituz, gailu xumeagoetan sartzeko diseinatu dena.

1.2 Dokumentuaren antolakuntza

Dokumentu hau sarrera kapitulu honekin hasten da. Ondoren, 2. eta 3. kapituluetan, proiektua nola antolatu eta planifikatu den deskribatzen da eta gero aurretik gai honen inguruan egindako lana aztertu da ere. 4. kapituluan CoAP deskribatzen da, bere espezifikazioa eta baita gehigarri ezberdinak, garatzaileentzat libre erabiltzeko dauden inplementazio batzuk aurkeztearekin batera. Hurrengo kapituluan inplementazio hauen arteko konparaketa teorikoa egiten da, inplementazio bakoitzaren ezaugarriak analiatuz. Behin analisi teorikoa eginda, analisi praktikoa egin da 6. eta 7 kapituluetan. 6. experimentua deskribatzen da eta 7. beraien emaitzak. Bukatzeko, proiektuaren ondorioak azaltzen dira, bai ondorio teknikoak eta baita planifikazioaren ingurukoak, gai honen inguruan lan egiten jarraitzeko ideia batzukekin batera.

2. KAPITULUA

Proiektuaren Helburuak eta Planifikazioa

Gradu bukaerako proiektu honen helburua CoAP inplementazio ezberdinak konparatzea da. Horretarako, hasieran hainbat inplementazio modu teorikoan aztertzen dira, haien ezaugarriak deskribatuz eta non izan daitekeen interesgarriagoa inplementazio bakoitza erabiltzea. Ondoren, Raspberry Pi plataforma batean martxan jarri dira inplementazio horiek eta hainbat neurri hartu dira, konparaketa praktiko bat ere edukitzeko. Analisi eta konparaketa bai praktiko eta teorikoari esker, sistema diseinatzaileei beraien lana erraztea da lan honen helburua, behin CoAP aukeratu eta gero, inplementazio egokiena hautatzen lagunduta.

2.1 Azpiatazen zerrenda

Aurreko helburuak aurrera eramateko, hainbat ataza definitu dira. Lehenik liburutegiak ezagutu eta teorikoki analizatuko dira. Ondoren, inplementazioak martxan jarri eta demoak prestatuko dira. Alde praktikoarekin bukatzeko, froga ezberdinak egin eta hainbat neurri hartuko dira. Proiektua bukatzeko memoria hau idazteaz gain artikulu zientifiko bat ere idatziko da.

1. CoAP liburutegien gaur egungo egoeraren analisia egin.
2. CoAP-ek beste protokolo arin batzuekiko eskeintzen dituen abantailak eta desabantailak identifikatu.

3. CoAP liburutegien konparaketa teorikoa egin.
4. CoAP liburutegi ezberdinekin demoak prestatu.
5. CoAP liburutegien arteko elkarreragingarritasun aztertu.
6. CoAP liburutegien analisi praktikoa egin.
7. Egindako lanetik artikulu zientifiko bat prestatu.
8. Memoria idatzi

2.2 Planifikazioa

Planifikazioarekin jarraituz, atal honetan atazak eta egutegia definitzen dira.

2.2.1 Denboraren estimazioa

Denborari dagokionez 2.1 taulan azaltzen da ataza eta azpiataza bakointzarentzako planifikatutako denbora.

Ataza	**Denbora**
Guztira	300
Prozesu taktikoak	15
K – Kudeaketa	13
K1 – Bilerak	8
K2 – Artxiboaren kudeaketa	5
P – Planifikazioa	2
P1 – PHDa egin	2
Prozesu operatiboak	285
G – Garapena	185
G1 – Aurrekariak analizatu	50
G2 – Liburutegiak detektatu eta probatu	65
G2.1 - Detektatu	15
G2.2 – Teorikoki analizatu	25
G2.3 – Probatu	25
G2 – Demoak prestatu	50
G3 – Neurketak egin	20
G3.1 – Interoperabilitatea probatu	5
G3.2 – Latentziak neurtu	5
G3.3 – RAM eta CPU-a neurtu	5
G3.4 – ROM kontsumoa neurtu	5
D – Dokumentazioa	100
D1 – Artikulua idatzi	50
D2 – Memoria idatzi	50

2.1 Taula: Atazen denbora

2.2.2 Denbora plangintza: GANTT Diagrama

2.1 irudian Gantt diagrama erakusten da, ataza bakoitza noiz hasi eta noiz bukatzen den adierazten duena.

2.1 Irudia: Gantt diagrama

2.3 Arriskuak

Proiektua aurrera eramaterakoan hainbat arazo sor daitzezke planifikazioa izorratu ditzaketenak. Gauzak horrela izanda, ondorengo arriksuak detektatu eta definitu dira:

A01-Lanaren atzerapena.

Deskribapena: Lana arrazoi desberdinen ondorioz, finkatu den epera iristeko zailtasuna edukitzea.
Larritasun-maila: ALTUA. Epeak ez direlako errespetatuko.
Eragina: Finkatu diren epeak ez errespetatzea.
Aurrezaintza-plana: Disziplinatua izatea eta dena egin beharreko epean egitea.
Kontingentzia-plana: Epea baino lehen beste epe bat jarri nire buruari eta ordurako bukatuta eduki.

A02-Ezagutza falta.

Deskribapena: Proiektuan erabiltzen diren programak edo teknologia erabiltzeko arazoak.
Larritasun-maila: ERTAINA. Lana geldotzea.
Eragina: Lana geldotzen da.
Aurrezaintza-plana: Lana egiteko teknologia jakinda, hasi baino lehen jakinaren gainean jarri eta ikasi.
Kontigentzia-plana: Lehen bait lehen egunean jarri jakin beharrekoarekin.

A03- Bat-bateko lan karga

Deskribapena: Beste alde batetik lana jasotzea.
Larritasun-maila: Aldakorra, eskaera kopuru eta epeen arabera.
Eragina: Lanaren atzerapena eta lana pilatzea.
Aurrezaintza-plana: Lanak planifikazioan estimatutakoa baino lehenago bukatzea.
Kontingentzia-plana: Lan gehiago egitea eta estresatu beharrean lanak banan banan egiten joatea.

A04- Datuen galera

Deskribapena: Emangarri zehatz baten, batzuen zein artxibo osoa desagertzea.
Larritasun-maila: Aldakorra, galdurako bolumenaren arabera.

Eragina: Lana errepikatu behar izatea.
Aurrezaintza-plana: Artxiboa interneten izateaz gain segurtasun kopiak egitea.
Kontingentzia-plana: Ordu estrak dedikatu galdutako lana berregiten, unekoa alde batera utzi gabe.

3. KAPITULUA

IoT eta CoAP, Aurrekariak

IoT eguneroko objektu edo gauza fisikoak interkonektatzen dituen sarea da, zirkuitu elektronikoak, sentsoreak, softwarea eta sare gaitasuna txertatu zaizkienak, datuak elkarbanatu eta jasotzea ahalbideratzen duena. Era honetan, hurrunetik informazioa jaso edo objektuak kontrola daitezke dagoeneko existitzen den sare azpiegitura bat erabiliz. Horrela izanda, adituek espero dute 2020-rako 50 mila miloi gailu egongo direla internetera konektatua [Evans, 2011], [Chase, 2013]. Ziurrenik gehiegi izango dira, 2015-an jada aurreikuspenen atzetik baitzihoan konektatutako gailuen zenbakia [Evans, 2011], baina egia da egunero gailu berriak konektatzen direla internetera.

Gailu hauek ez dira ordenagailu arruntak bezain indartsuak eta energia kontsumo baxuagua beharko dute, bateriekin edo pilekin funtzionatuko baitute. Hori dela eta, IoT-ra zuzendatutako protokolo berriak agertzen joan dira eta horietako bat lan honetan aurkezten den CoAP da.

Badaude ikerketak non, CoAP-en eraginkortasuna aztertzen den, inplementazio ezberdinak eta hauen ezaugarrien deskribapena. Hasieran, CoAP-en estandarizazioa oraindik burutu gabe zegoenean, inplementazio batzuen analisia egin zen artikulu batzuetan: [Lerche et al., 2012] eta [Villaverde et al., 2012]. Bigarrenean, CoAP teorikoki analizatzen da, CoAP-en definizioaren lehen zirriborroa aurkeztu eta denbora gutxira, orduko inplementazio batzuekin batera. Zeuden inplementazioak aurkezten ditu eta beste artikuluetatik ateratako analisiekin ondorio batzuk ateratzen dira bertan, baina analisi propiorik egin gabe. [Lerche et al., 2012]-k, ETSI CoAP Plugtest [ETSI, 2012]-eko emaitzak aurkezten dituzte. Bertan erabilitako liburutegiak aurkezten dira eta zein test pasatu zuten, baina ez dute eraginkortasunik az-

tertzen. Bi artikuluek aurkezten dute inplementazioen zerrenda bat, baina idatzi zirenetik egoera aldatu egin da, CoAP-en estandarizazio prozesua bukatu da eta inplementazio berriak agertu eta zahar batzuk desagertu egin dira. CoAP-en eraginkortasunaren inguruan informazioa aurkitzeko bilaketa parametroak zabaldu behar dira, beste protokolo batzuekiko konparaketa edo hardware ezberdinaren gainean egindako probak kontutan edukitzeko.

CoAP-en liburutegi bakar bat analizatzen dutenen artean, [Ludovici et al., 2013]-ek beraien inplementazio propioa aurkezten dute, TinyOS plataformarako: TinyCoAP. Eraginkortasuna konparatzen dute HTTP/TCP, HTTP/UDP eta TinyOS-ren inplementazio originalaren, CoAPBlib-en artean. TelosB motak erabiltzen dituzte horretarako eta latentzia, memoria eta energia kontsumoa neurtuz. [Kruger and Hancke, 2014]-k benchmark bat azaltzen dute, CoAP-en inplementazio berdina Raspberry Pi, BeagleBone eta BeagleBone Black batean martxan jarrita. 4 eta 10 klaseko SD txartelak erabiltzen dituzte eta GUI-a martxan, itzalita eta desinstalatuta egiten dituzte neurketak. Latentzia, banda zabalera eta erabilitako CPU eta memoria dira neurtutako parametroak, guztiak loopback interfazean. Horrez gain, latentzia ere konparatzen dute TelosB mota batean dagoen zerbitzari batean eta BeagleBone bat gateway bezala erabilita.

Analizatutako protokoloak zabalduz gero, CoAP eta HTTP-k antzeko estruktura dute, nahiz eta inguru ezberdinetara orientatuta egon. [Colitti et al., 2011]-ek Tmote Sky eta Zolertia Z1 motekin lan egin dute, erantzun denbora eta energia kontsumoa neurtzeko. [Kuladinithi et al., 2011]-k aldiz, beraien CoAP inplementazioa erakusten dute, libcoap, eta Contiki eta TinyOS-ra portak egin ondoren HTTP-rekiko konparatzen dute. Azkenik, [Elmangoush et al., 2015]-k CoAP, HTTP, MQTT eta AMQP azaltzen dituzte, baina lehenengo biekin bakarrik egiten dute lan. OpenMTC plataformaren gainean banda zabalera eta erantzun denbora neurtuz.

MQTT da beste IoT-rako protokolo oso ezagun bat eta [De Caro et al., 2013]-k hainbat parametro neurtzen dituzte MQTT eta CoAP-ekin. Latentzia, banda zabalera eta paketeen galera ratioa neurtzen dute zerbitzu kalitate eta sare galera konfigurazio ezberdinetan, Smartphone batean inplemetatuta. [Thangavel et al., 2014]-k, bestalde, middleware komun bat aurkezten dute, MQTT eta CoAP-entzako eta latentzia eta banda zabalera neurtu.

Beste hainbat protokolo ere erabiltzen dira IoT-rako. Horren arira, [Talaminos-Barroso et al., 2016]-k eHealth herramienta bat inplementatu dute, DDS, MQTT, CoAP, JMS, AMQP eta XMPP erabiltzeko aukera ematen duena. CPU eta memoria erabilera, banda zabalera eta

mezuen latentzia eta jitter-a neurtzen dituzte. [Mun et al., 2016]-k, aldiz CoAP, MQTT, MQTT-SN, WebSockets eta TCP aukeratu dituzte analizatzeko. Aplikazio garatzaileei protokoloa aukeratzen laguntzea da haien helburua eta horretarako energia eraginkortasuna eta memoria eta CPU erabilera neurtzen dituzte. Bukatzeko, [Chen and Kunz, 2016]-k MQTT, CoAP, DDS eta UDP-ren gainetik doan protokolo propio bat analizatzen dute. Sare emuladore bat erabiltzen dute sareko hainbat parametro konfiguratu eta banda zabalera, latentzia eta pakete galera neurtzen dituzte.

Aurreko parrafoetan azaldu bezala, CoAP-en eraginkortasuna aurretik ere konparatu da, beste hainbat protokolorekiko. Inplementazio konkretu batzuen eraginkortasuna ere analizatu da, hardware ezberdinetan. Hala eta guztiz ere, egindako lan guztia CoAP-en definizioaren hasierako zirriborroak inplementatzen dituzten liburutegietan oinarrituta dago, beraz ez dago eguneratuta. Gainera, gaitasunak eta interoperabilitatea konparatzen dituzte eta ez dute eraginkortasuna aztertzen. Ondorioz, lan honen helburu nagusietako bat da hutsune hori betetzea, gaur egungo konparaketa bat egin eta teorikoki eta praktikoki aztertu, era honetan, sistema diseinatzaileei behar duten inplementazioa aukeratzen lagunduz.

4. KAPITULUA

Constrained Application Protocol (CoAP)

IoT-n indar gehien hartzen ari diren protokoloak dira CoAP eta MQTT. Nahiz eta MQTT helduagoa den, ikusi da erabilpen kasu batzuetarako ez dela egokia, beraz, CoAP erabiltzen hasi dira garatzaileak. Biek arkitektura ezberdina daukate, beraz, bakoitzak badauka bere lekua IoT-ko ekosistema heterogeneoan. MQTT-k argitaratzaile/harpide eredua jarraitzen du eta TCP-ren gainean doa TCP pilan eta CoAP-ek bezero/zerbitzari eredua UDP-ren gainean. Kontrol aplikazioetarako eta 1-1 komunikazioetarako egokiagoa da CoAP, eta beraz, garrantzitsua da horrelako aplikazio bat garatzerako orduan, behin CoAP erabiliko dela erabakitakoan, inplementazio egokia aukeratzea.

Constrained Application Protocol (CoAP) [Shelby et al., 2014] gailu elektroniko txikietan erabiltzeko diseinatu den software protokolo bat da, internet bidez komunikatzeko gaitzen duena. Oso energia gutxi kontsumitzen duten gailuetara orientatuta dago batez ere, adibidez sentsoreak, etengailuak, balbulak eta antzeko osagaietarako, kontrolatu edo gainbegiratuak izateko beharra daukatenak. HTTP-ren diseinua jarraituz, REST paradigma jarraitzen du eta oso erraz bihurtu daiteke HTTP proxy simpleen bidez egungo sareetan integrazioa errazteko, baina beste ezaugarri batzuk gehitzen dizkio, hala nola, multicast, kontsumo baxua eta arintasuna. Hiru ezaugarri hauek oso garrantzitsuak dira IoT eta Machine-to-Machine komunikazioetarako, gailu hauek sistema txikietan txertatuta egoten baitira eta memoria eta energia mugatuak izaten baitituzte, eraginkortasuna oso garrantzitsua delarik. CoAP UDP edo antzeko protokolo bat onartzen duten gailu gehienetan erabil daiteke.

The Internet Engineering Task Force (IETF)-ko Constrained Restful Environments (Co-

RE) lantaldea izan da CoAP-en estandarizazio lana egin duena eta horrela aurkeztu zuten 2014-an RFC 7252 kodearekin. RFC horretan oinarria dago eta bera osatzeko hainbat gehigarri estandarizazio prozesuko etapa ezberdinetan daude.

4.1 Ezaugarriak

CoRE lantaldeak ondorengo ezaugarriak kontutan edukita diseinatu du CoAP

- Analisi eta kontsumoaren konplexutasuna.
- URI eta content-type onartu beharra.
- Ezagunak diren CoAP zerbitzuek dauzkaten baliabideen aurkikuntza onartu beharra.
- Baliabideetara erraz harpidetzea, push notifikazioak jaso ahal izateko.
- Katxeatu sinplea, max-age-n oinarrituta.

HTTP-ra mapeoa definituta dago, proxy-ak erraz sortzea ahalbidetuz, CoAP baliabideetara HTTP bidez sarbidea eman daitekeelarik. Behin CoAP definituta, protokolo libre ugari daude konexioen pila osorako, baita inguru eta gailu mugatuetarako. CoAP erabiltzean, protokoloen pila ezberdina izaten da interneteko aplikazio orokorretarako, 4.1 taulan ikus daitekeen bezala

Maila	**IoT**	**Aplikazio orokorrak**
Aplikazioa	CoAP, CoAPs	HTTP, HTTPs
Garraioaren segurtasuna	DTLS	TLS
Garraioa	UDP	TCP
Sarea	IPv6	IPv6, IPv4
	6LoWPAN	
Lotura	IEEE 802.15.4	IEEE 802.3, 802.11

4.1 Taula: Protokoloen pila

4.1.1 Zerbitzuaren kalitatea

Zerbitzuaren kalitateak definitzen du ea konprobatzen den mezua hartu den edo ez, hau da, mezua jasotzen ez denean zer egin behar den. UDP-k ez du inongo konprobaketarik egiten, ez du inongo segurtasunik ematen ea mezua iritsi den ala ez. Hori dela eta, CoAP da konprobaketa hori egiten duena. Horretarako, bi zerbitzu kalitate eskeintzen ditu, mezu Confirmable-ak eta Non-Confirmable-ak. Lehenengoaren kasuan, mezua bidali eta gero ACK baten zain geratuko da aplikazioa, eta heltzen ez bada berriro bidaliko du mezua. Bigarren kasuan, mezua bidali eta aplikazioak aurrera jarraituko du, ez du inongo konprobaketarik egingo.

4.1.2 Segurtasuna

Komunikazioan segurtasuna gehitu bada, interneteko aplikazio orokorretan normalean TLS erabiltzen da. CoAP-en kasuan hau ez da posible, TLS-k mezuan helduko direla, eta gainera ordenean egingo dutela behar du, eta UDP-k ez dauka ezaugarri hau. Hori dela eta, Datagram Trasnport Layer Security (DTLS) diseinatu da, UDP-ren gainean TLS-ren teknikak aplikatuz. Hainbat protokolo bakar batean biltzen ditu:

- Handshake protokoloa: Gakoen elkarbanatzea eta konexio iraunkor baten konfigurazioa nola egin definitzen du.
- Record protokoloa: Komunikazio segurua definitzen du.
- Alert protokoloa: Erroreen notifikazioa definitzen du.
- ChangeCipherSpec protokoloa: Mezu bakar bat dauka, logikoki handshake prozesuaren barruan dago, baina protokolo ezberdin bat bezala hartzen da.

Hala ere, DTLS-k hainbat ezaugarri gehitzen dizkio komunikazioari eta honek UDP-k TCP-rekiko dauzkan abantaila batzuk galaraztea ekar dezake, hala nola azkartasuna eta konexioa zabalik eduki behar ez izatea.

Segurtasun mailari dagokionez, NoSec edo segurtasunik gabekoaz gain, beste hiru segurtasun maila definitzen dira:

- Aurretik partekatutako gakoak: Gako simetrikoak ematen zaizkio bai bezero eta baita zerbitzariari, beraz, kriptografia simetrikoan oinarritzen da.

- Gako publikodun ziurtagiri gordinak: Kriptografia asimetrikoko algoritmoak erabiltzen dira, baina ziurtagirietan gakoak bakarrik daude.
- X.509 ziurtagiriak: Aurreko kasuaren antzekoa da, gakoak gordetzeko eta elkarbanatzeko era aldatzen da, X.509 formatua erabiltzen dute ziurtagiri hauek.

4.2 CoAP-en Espezifikazioa

4.2.1 RFC 7252

RFC 7252 da erreferentzia iraunkorra. 2013ko Uztailaren 11n onartu zen textitdraft-ietf-core-coap-18 izenarekin eta 2014ko Ekainaren 26an RFC 7252 publikatu zen. Atzerapenaren arrazoia segurtasun espezifikazio batzuen publikazio prozesua bukatzea izan zen.

4.2.2 IANA erregistroa

CoAP hedagarria izateko diseinatua izan da. Hori dela eta, hainbat erregistro definitzen dira protokoloaren bilakaerari laguntzeko. Erantzun kodeak (adibidez 4.04 “Not found”) eta edukiaren formatua identifikatzeko kodeak (50 application/json-rako) IANA erregistroan jasotzen dira. Era honetan espezifikazioak guztiontzat ikusgarri geratzen da eta bermatzen da kodeak ez direla elkar zapalduko.

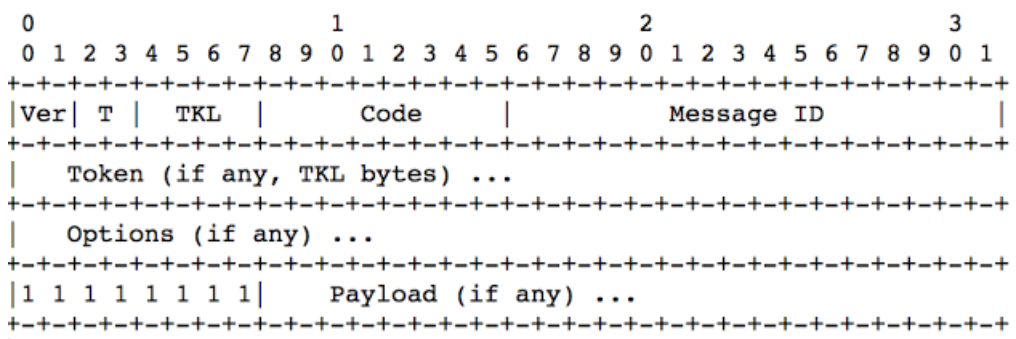
4.2.3 Mezuen formatua

CoAP-ek bi mezu mota erabiltzen ditu, eskaerak eta erantzunak. Horretarako, formatu bitar sinple bat erabiltzen du goiburuetarako. Oinarrizko goiburuari aukera ezberdinak gehitu ahal zaizkio, formatu eraginkor batekin (Luzeera-Balioa).

4.1 irudian ikus daitezke CoAP mezu batean dagoen zati bakoitza.

Goiburuaren atzetik datorren edozein byte, mezuaren edukia izango da. Mezuaren luzeera datagramaren luzeeran inplizituki doa. IETF-ren gomendioa datagrama bakar batean mezu osoa sartzea da, mezuen fragmentazioa ekiditeko.

- Lehenengo 2 bit-ak (Ver) CoAP-en bertsioari dagozkio.



4.1 Irudia: CoAP mezu baten goiburua

- 2 bit-ek (T) mezu mota definitzen dute. Confirmable, Non-Confirmable, ACK edo Reset.
- 4 bit (TKL) tokenaren luzeerarako. Token-a ez denez beharrezkoa lau 0-ko izan daitezke.
- 8 biteko kodea: Aurrerago azaltzen dira.
- 16 bit: mezuaren ID-rako.

Hauen atzetik datozen goiburuko bit-ak hautazkoak dira, goiburua eta edukia banatzen duten eta 1-ra jarrita dauden zortzi bitak izan ezik. Azken hau mezuan edukia baldin badago beharrezkoa da, bestela ez da jartzen. Era honetan, CoAP mezu baten tamaina minimoa 32 bit edo 4 bytekoa da.

Mezuen kodeak hauen motaren menpe daude. X.YY formatuan erakusten dira, non X-ak klasea adierazten duen eta Y-ek mota konkretuagoa. Eskaera batean, X-ek 0 balioa edukiko du eta erantzunak datuen jasoera ondo joan den edo ez edukiko du kontutan. 4.2, 4.3 eta 4.4 tauletan ageri dira kode posibleak, bai eskaera eta baita erantzunetarako ere.

Kodea	Izena
0.00	EMPTY
0.01	GET
0.02	POST
0.03	PUT
0.04	DELETE

4.2 Taula: Eskaeren kodeak

Klasea	Deskribapena
2.xx	Ondo
4.xx	Bezeroaren errorea
5.xx	Zerbitzariaren errorea

4.3 Taula: Erantzunen kodeak

2.YY motako erantzunek eskaera ondo exekutatu dela adierazten dute, 4.YY motakoek bezeroaren eskaeran erroreren bat dagoela eta 5.YY zerbitzariak arazoren bat duela.

Kodea	Deskribapena
2.01 (65)	Created
2.02 (66)	Deleted
2.03 (67)	Valid
2.04 (68)	Changed
2.05 (69)	Content
4.00 (128)	Bad Request
4.01 (129)	Unauthorized
4.02 (130)	Bad Option
4.03 (131)	Forbidden
4.04 (132)	Not Found
4.05 (133)	Method Not Allowed
4.06 (134)	Not Acceptable
4.12 (140)	Precondition Failed
4.13 (141)	Request Entity Too Large
4.15 (143)	Unsupported Content-Format
5.00 (160)	Internal Server Error
5.01 (161)	Not Implemented
5.02 (162)	Bad Gateway
5.03 (163)	Service Unavailable
5.04 (164)	Gateway Timeout
5.05 (165)	Proxying Not Supported

4.4 Taula: Erantzunen kodeak

Edukia edo payload barruan doazen balioek formatu ezberdinak eduki ditzakete eta 4.5 taulan erakusten dira.

Media type	Id.
text/plain;charset=utf-8	0
application/link-format	40
application/xml	41
application/octet-stream	42
application/exi	47
application/json	50
application/cbor	60

4.5 Taula: Edukien formatua

4.2.4 Gehigarriak

IETF-k gehigarri batzuk proposatzen ditu CoAP-en ezaugarriak zabaltzeko. Hauetako batzuek CoAP-en ezaugarriak erabiltzea errazten dute eta besteek zabaldu, aplikazio konplexuagoak ahalbidetuz.

- **Constrained RESTful Environments (CoRE) Link Format** [Shelby, 2012]: Gehigarri honekin, IETF-k zerbitzari mugatuen linken formatua definitzen du: baliabideak, atributuak eta linken arteko harremanak.
- **Block-Wise Transfers in the Constrained Application Protocol (CoAP)** [Bormann and Shelby, 2 eduki handia duten mezuak bidaltzeko definitu da gehigarri hau, IP fragmentazioa ekiditeko. Hau gerta daiteke firmware eguneraketetan, adibidez.
- **CoRE Resource Directory** [Shelby et al., 2016]: IETF-k Resource Directory izeneko entitate bat proposatzen du, beste CoAP zerbitzari batzuetako baliabideei buruzko informazioa gordetzen duena. Era honetan, zerbitzari horiek energia aurrezten dute eta zerbitzariak bilatzea erraz daiteke.
- **Observing Resources in the Constrained Application Protocol (CoAP)** [Hartke, 2015]: Gehigarri honekin, CoAP zerbitzariak gai dira harpidetuta dauden bezeroei push notifikazioak bidaltzeko, beste argitaratzaile/harpide eredua jarraitzen duten protokoloen antzera, adibidez MQTT edo XMPP.

- **Group Communication for the Constrained Application Protocol (CoAP)** [Rahman and Dijk, 2014]: multicast inguru batean CoAP nola erabili definitzen du gehigarri honek.

- **CoAP Simple Congestion Control/Advanced CoCoA** [Bormann et al., 2015]: CoAP-en espezifikazioak kongestioa kontrolatzeko metodo sinpleak proposatzen ditu, gehigarri honekin metodo konplexuak ahalbidetzen dira.

4.2.5 Inplementazioak

CoAP inplementazio libre ugari daude eskuragarri, ezaugarri ezberdinak dauzkatenak eta plataforma ezberdinetara orientatutakoak. Lan honen helburua inplementazio horietako batzuk konparatzea da, plataforma berean. Lengoaia eta exekuzio ingurune ezberdinetara orientatutako liburutegiak aukeratu dira, eta ondokoak izan dira aukeratutakoak:

- **libcoap** [Bergmann, 2016] C-ko liburutegi bat da, gailu txertatu txikietatik POSIX SE duten sistema handietara doazen gailuetara bideratutakoa. RFC 7275 ofiziala jarraitzen du, bezero eta zerbitzari aldea inplementatzeko laguntza eskeintzen du eta hainbat gehigarri ere baditu: Observe mode, Block-wise transfer eta Resource Directory. Kodea adibide oso konpletuekin dago argitaratuta eta DTLS segurtasuna OpenSSL edo tinyDTLS-rekin gehitzeko erraztasunak emateko dago diseinatuta.

- **smcp** [Quattlebaum, 2016] ere C-n inplementatuta dago eta libcoap-en antzera, gailu txiki eta handietan sartzeko pentsatuta dago. Sistema eragilerik gabeko sentsore txikietatik, GNU/Linux daukaten gailu handietara, sistema txertatuak barne. Bai bezero eta bai zerbitzariak inplementatzeko laguntzak ematen ditu eta RFC 7252-a jarraitzen du. Gehigarriei dagokienez, Observe mode eta Multicast taldeak onartzen ditu. Komando lerroko bezero bat ematen du, *smcpctl*, eta egun experimentala den DTLS adar bat ere bai.

- **microcoap** [1248, 2016] mikrokontrolagailu txikietan sartzeko C liburutegia bat da. POSIX eta Arduino-rako adibideak eskaintzen ditu kodearekin batera, eta RFC 7252 jarraitzen du, baina ez du osorik inplementatzen. Zerbitzariaren aldea bakarrik eskaintzen du eta hau ere mugatuta dago, GET, PUT eta POST eskaeretara, eta erantzunak ACK-ren barnean bakarrik bueltatzen ditu. Gainera ez du birtransmisiorik egiten.

- **FreeCoAP** [Cullen, 2016] da lan honetan aztertutako azken C liburutegia. GNU/Linux gailuentzako diseinatuta dago, GnuTLS erabiltzen baitu segurtasunerako. RFC 7252 ofiziala jarraitzen du.

- **Californium** [The Eclipse Foundation, 2014] Java liburutegi bat da, oso konpletua baina ezin da gailu mugatuetan sartu. Backend sistemetarako dago orientatuta, JVM erabiliz eta bezero eta zerbitzariak inplementatzeko laguntza ematen du. RFC 7252-z gain, Observe mode, Block-wise transfer eta Resource Directory dauzka eta DTLS segurtasuna Scandium proiektuarekin gehitu dakioke.

- **h5.coap** [morkai, 2014] Node.js-rako JavaScript liburutegia da. Bezeroak inplementatzeko bakarrik balio du eta RFC 7252-a jarraitzen du. Gainera, Observe mode eta Block-wise transfer gehigarriak ere inplementatuta dauzka.

- **node-coap** [mccollina, 2016] era JavaScripten idatzita dago eta Node.js-rako diseinatu da. h5.coap-ek ez bezala, honek bai bezero, baita zerbitzariak inplementatzeko balio du eta RFC 7252-ren bertsio iraunkorra jarraintzen du. RFC-az gain, Observe mode eta Block-wise transfer gehigarriak ere badauzka inplementatuta.

- **CoAPthon** [Tanganelli, 2016] Pythonentzako liburutegia da. Bezero eta zerbitzeriak inplementa daitezke eta RFC 7252-a jarraitzen du. Horrez gain, Observe mode, Core-Link format, Multicast eta Block-wise transfer gehigarriak ere inplementatzen ditu.

- **CoAPy** [CoAPy, 2010] ere Python liburutegi bat da, baina kasu honetan espezifikazioaren zirriborro zahar batean oinarrituta dago, draft-ietf-core-coap-02. Era honetan, teorikoki ez luke beste inplementazioekin interoperablea izan behar. Zirriborroaz gain, Block-wise transfer gehigarria ere inplementatzen du.

Aurretik esan bezala, CoAP inplementazio gehiago ere badaude, baina aurreko zerrendak ondo errepresentatu ditzake lengoaia eta exekuzio ingurune ezberdinak. Hala ere, merezi du beste batzuk ere aipatzea, arrazoi ezberdinengatik alde batera utzi direnak:

- **Copper** [Kovatsch, 2014] Firefox-erako plugin bat da, oso erabilgarria inplementatutako zerbitzariak frogatzeko. RFC 7252 jarraitzen du eta Resource Directory, Block-wise transfers eta Observe mode gehigarriak ere inplementatzen ditu, gizakiok ikusteko eran. Interoperabilitatea probatzeko erabili da, baina ez da neurririk hartu Copper-ekin.

- **Erbium** [ETH Zurich, 2014] C-n inplementatuta dago eta oso erabilia da, baina ContikiOS plataformarako orientatuta. Lan onetan Raspberry Pi-ekin lan egingo denez, alde batera utzi da.
- **TinyCoAP** [AleLudovici, 2013]-rekin Erbium-en gauza bera gertatzen da, oso erabilia da, C-n inplementatuta dago baina tinyOS-rako diseinatu da.

5. KAPITULUA

Inplementazioen analisia eta konparaketa teorikoa

Aurreko kapituluan CoAP protokoloa bera azaldu da eta hainbat ezaugarri aurkeztu dira. Kapitulu honetan liburutegien konparaketa teorikoa egingo da. Horretarako, inplementazioen ezaugarriak azalduko dira (lengoaia, non inplementatzeko diseinatu diren, dauzkaten gehigarriak...) eta ondoren zein erraztasun ematen duten zerbitzariak martxan jartzeko, hau da, zerbait berezia egin behar den eskaerak tratatzeko edo ez.

5.1 taulak analizatutako liburutegien ezaugarriak azaltzen ditu. Zein bertsio probatu den, zein plataformatan sartzeko diseinatu den, programazio lenguaia eta oharrak. Taulan azaltzen den bezala, lau C inplementazio daude, hainbat plataformatarako eta horrez gain, Java inplementazio bat, Javascripterako bi eta beste bi Pythonen. Taulako azkena, Copper, interoperabilitaterako soilik erabili da. Firefox-entzako plugin bat da, zerbitzariak probatzeko oso erabilgarria dena, era oso ikus-erraz batean zerbitzarien baliabideak ikusteko balio duena.

Bestalde, aipatzekoa da microcoap, RFC 7252 ofiziala jarraitu arren, ez du osorik inplementatzen eta ezaugarri batzuk faltan dauzka. GET, PUT eta POST eskaerak onartzen ditu, DELETE-ak ez. Gainera, ez du birtransmisiorik onartzen eta beti bueltatzen du ACK bat, nahiz eta eskaerak ez konfirmagarri bezala egin. Gainera, erantzuterakoan edukia beti ACK-n joango da, *piggibacking* deritzon tekina jarraituz.

CoAPy-ren kasua ere komentatzekoa da. Informazioa bilatzerako orduan toki askotan azaltzen da, baina proiektua alde batera utzi dute garatzaileek. Hala ere, lan honetarako kontuan eduki da, CoAP espezifikazio zahar bat jarraitzen duelako, eta bera, bertsio ezberdinen arteko komunikazioa aztertu ahal izateko.

Liburutegia	Bertsioa	Lengoaia	Plataforma	Oharrak
libcoap	Develop - 2016 Irailak 24	C	POSIX, Contiki, lwIP, TinyOS	-
smcp	Master - 2016 Irailak 24	C	Gailu txertatuak,bare-metal sentsoreak, Linux gailuak	-
microcoap	Master - 2016 Irailak 24	C	Arduino, POSIX	Zerbitzaria bakarrik, RFC-a partzialki
FreeCoAP	Master - 2016 Irailak 24	C	GNU/Linux	-
Californium	1.1.0-SNAPSHOT	Java	JVM	-
h5.coap	0.0.0	JavaScript	Node.js	Bezeroa bakarrik
node-coap	0.18.0	JavaScript	Node,js	-
CoAPthon	Master - 2016 Irailak 24	Python	Python	-
CoAPy	0.0.3-DEV	Python	Python	Abandonatua
Copper	1.0.0	GUI	Firefox	Bezeroa bakarrik, Firefox plugin-a

5.1 Taula: Inplementazioen laburpena

5.1 taulan ezaugarri teknikoagoak erakusten dira. Ikus daitekenez, liburutegi guztiek RFC7252-a jarraitzen dute, CoAPy-k izan ezik, 2. zirriborroa jarraitzen du. Gehienek bai bezeroak eta baita zerbitzariak ere inplementatzeko balio dute, baina ez denek. Copper bezero bat da, ez da liburutegi bat, h5.coap-ek bezeroak bakarrik inplementatzeko balio du eta microcoap-ek zerbitzarientzako soilik. Gehigarrien artean, nahiz eta gehienek esan ez, CoRE Link Format badaukate. Bestalde Obseve mode eta Block-wise transfer erabiltzeko gai dira gehienak, gehigarri oso erabilgarri eta gainera nahiko denbora daukatenak baitira. Liburutegi helduenek (libcoap eta Californium), gainera Resource Directory ere badaukate programatuta.

Eraginkortasunari dagokionez, C inplementazioek beharko lukete azkarrenak eta pisu gutxienekoak, teorikoki behintzat. C makina kodera konpilatzen delako da hori, hemen aztertutako beste lengoaiek beste kapa bat behar baitute sistema eragilearen gainean: Javak JVM behar du, Python-ek bere interpretatzailea eta JavaScripteko inplementaziek Node.js exekuzio ingurunea.

Aztertutako inplementazio guztiek, CoAPy-k izan ezik, esaten dute RFC 7252 jarraitzen dutela, beraz, elkarrekin ondo komunikatu beharko lukete, hau da, interoperableak izan beharko lukete. CoAPy-k draft-02 zirriborroa jarraitzen du, eta Path-Uri aukera aldatu egin da, lehen 9 kodea erabiltzen zuen eta draft-03-tik aurrera aukera hori hainbatetan banatu da. Gainera, ez du 9 kodea erabiltzen zati horietako batek ere ez, beraz ez dago konpatibilitaterik haien artean, 7.1 taulan ikus daitekeen bezala. Beste ezberdintasun bat da PUT-en erantzunak tratatzeko era. Lan honetan PUT-ei mezu batekin erantzun behar zaie eta erantzun egokiak 2.04 (Changed) kodea eduki beharko luke RFC 7252-a jarraituz,

baina draft-02-ak ez dauka kode hau. Kasu honetan, 2.01 (OK) kodea erabiltzea erabaki da, GET-entzakoa dena.

Bestalde, lan hau burutzean, CoAPThon-en arazo bat aurkitu da: ez du aukerarik PUT edo POST baten erantzunean edukirik sartzen [Github, 2016] 2.04 (Changed) kodea duen mezu batean. Hau horrela izanda, kontutan eduki behar da honen erantzuna byte bat txikiagoa izango dela, denbora eta eraginkortasuna irabaziz.

Liburutegia	Espezifikazioa	Bezero/Zerbitzari	Gehigarriak
libcoap	RFC7252	Bezero-Zerbitzari	Observe, Block-wise, Resource directory
smcp	RFC7252	Bezero-Zerbitzari	Observe, Multicast
microcoap	RFC7252	Zerbitzari	-
FreeCoAP	RFC7252	Bezero-Zerbitzari	-
Californium	RFC7252	Bezero-Zerbitzari	Observe, Blockwise. Resource Directory
h5.coap	RFC7252	Bezero	Observe, Block-wise
node-coap	RFC7252	Bezero-Zerbitzari	Observe, Block-wise
CoAPthon	RFC7252	Bezero-Zerbitzari	Observe, Core-Link, Multicast, Block-wise
CoAPy	Draft-2	Bezero-Zerbitzari	Block-wise
Copper	RFC7252	Bezero	Observe, Block-wise, Resource Discovery

5.2 Taula: Inplementazioen ezaugarriak

* Nahiz eta gehienek beren deskribapenetan esan ez, Core Link-Format gehigarria ere inplementatzen dute.

Inplementazio batzuek beste batzuek baino ibilbide luzeagoa daukate eta horrek berarekin dakar funtzionalitateetan ez ezik, baliabideen, erantzun kodeen eta eskaera moten kudeaketan eboluzio bat eduki izana. Hori dela eta, baliabideak sisteman sartu eta kudeatzeko erak ezberdinak dira liburutegi bakoitzean eta honek eragin zuzena dauka aplikazioen garapen prozesuan, liburutegi batzuk beste batzuk baino askoz errazagoak dira erabiltzeko, ondoren deskribatzen den bezala:

- **libcoap**: interfaze bat dauka, baliabide berriak aplikazioan sartzea asko errazten duena. Liburutegiak berak kudeatzen ditu erantzun kodeak, garatzaileek baliabidearen izena, zein eskaera mota eskaitzen dituen eta eskaera mota bere handlerrarekin lotu behar dute, interfazearen bidez.
- **smcp**: libcoap-en antzera, garatzaileak handlerrak eta baliabideak sortu eta interfaze baten bidez sartzen ditu aplikazioan. Liburutegia arduratzen da erantzun kodeak eta gainerakoak kudeatzeaz.
- **microcoap**: baliabideak zerbitzarian sartzeko, definitu egin behar dira eta gero array batean sartu, handlerrekin batera. Gainerakoa liburutegiak kudeatzen du.

- **FreeCoAP**: erantzun kodeak aplikazioen garatzaileek kudeatu behar dituzte eta baita baliabideen path-a ere, liburutegiak ez dauka interfazerik automatikoki egiteko. Baliabideak sartzea eta erantzun kodeak kudeatzeak lana dakar, FreeCoAP-ek ez du era sinplerik eskeintzen honetarako.

- **Californium**: inplementazio hau oso heldua da eta erraztasun handiak eskaintzen ditu baliabideak sartu eta kudeatzeko. Aplikazioan baliabide berri bat sartzeko, Java klase bat sortu behar da, eskaera bakoitzaren handlerrarekin. Gero, interfaze baten bidez gehitzen da sistemara eta liburutegia arduratzen da gainontzeko guztiaz.

- **node-coap**: baliabideen eta erantzun kodeen kudeaketa garatzaileen esku geratzen da. Liburutegi honek ez du interfazerik eskaintzen honekin baliabideen sorkuntza eta kudeaketarako, beraz aplikazioak berak egin behar du.

- **CoAPthon**: Python klase bat sortu behar da baliabide bakoitzeko, aplikazioan kontutan eduki nahi diren metodoekin. Liburutegia arduratzen da erantzun kodeetaz eta gainerakoaz.

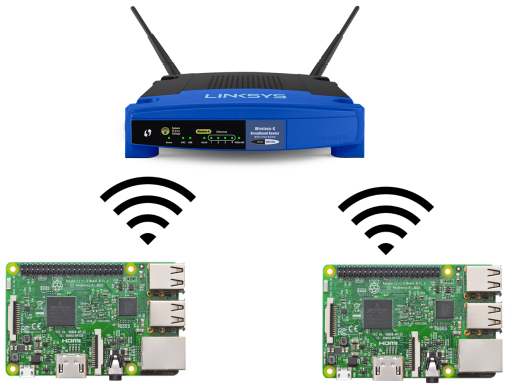
- **CoAPy**: kasu honetan ere Python klase bat sortu behar da baliabide bakoitzerako, baina aplikazioak parseatu behar du path-a eta erantzun kodeak kudeatu, liburutegiak ez du eskeintzen gaitasun hori.

Laburbilduz, libcoap, smcp, microcoap, Californium eta CoAPthon dira erabiltzeko liburutegi errazenak. Erantzun kodeak kudeatzen dituzte eta garatzaileek baliabideak definitu eta handlerrak esleitu behar dituzte, liburutegiak arduratzen dira gainontzekoaz, era garden batean.

6. KAPITULUA

Experimentuaren deskribapena

Egungo industrian sare kabledunak erabiltzen dira gehien bat, adibidez Profibus, Modbus, CAN, Profinet, etab. Sare hauen desabantaila nagusia da ez direla oso dinamikoak, hau da, gailu berri bat sartzeko edo dagoen bat lekuz aldatzeko kablea bota behar da. Hori dela eta, beste aukera kablerik gabeko sareak erabiltzea, beren fidagarritasuna hobetzen ari baita. Dokumentu honetan aurkezten diren experimentuetarako 2 Raspberry Pi 3 erabili dira, WiFi bidez konektatuta.



6.1 Irudia: Gailuen arteko konexioa

Raspberry Pi plataforma oso erabilia da prototipo industrialetarako eta gainera, lan honetarako daukan abantaila garrantzitsuena da C-s gain, beste lengoaietako inplementazioak era probatzeko balio duela. Gailu mugatuagoetan inplementatuz gero, seguruenik Python,

Java eta JavaScript inplementazioak kanpoan geratu beharko liratete. Gainera, gateway-ak inplementatzerako orduan kontutan edukitzeko plataforma da Raspberry Pi-a. Kasu honetan, bezero eta zerbitzari bat behar izan dira, beraz 2 Raspberry-ak erabili dira. Zerbitzarian led bat jarri da eskuragarri baliabide bezala eta bezeroak pultsadore bat dauka. Pultsadorea sakatutakoan PUT eskaera bat bidaltzen du zerbitzarira, 0 edo 1 edukiarekin, eta eduki horren arabera zerbitzariak led-a piztu edo itzali egingo du.

Esan bezala, 9 inplementazio probatu dira haien artean eta 10. bat, Copper, elkarren arteko interoperabilitatea probatzeko. Liburutegien kodea ikutu gabe utzi da, bezero eta zerbitzariak bakarrik programatu dira, kodeek eskeintzen duten adibideetan oinarrituta. Salbuespen bakarra FreeCoAP izan da, IPv6 bakarrik onartzen baitu eta IPv4 ere onartzeko port-ing bat egin behar izan zaio, sortutako sarea IPv4 izan baita.

Probetan, interoperabilitatea eta eraginkortasuna neurtu dira. Interoperabilitatearen kasuan, konbinazio posible guztiak probatu dira, hau da, liburutegi bakoitzaren bezeroa liburutegi guztien zerbitzari konektatzen saiatu eta erantzuten duen edo ez ikusi. Eraginkortasuna neurtzerakoan Copper kanpoan utzi da eta konbinazio guztian probatu dira. Kontutan eduki behar da h5.coap-ek ez duela zerbitzaririk eta microcoap-ek bezerorik sortzen.

Eskaerei dagokienez, byte 1-eko edukia daukate, 1 edo 0-ko bat, led-a piztu edo itzaltzeko eskatzeko. Erantzunetan, aldiz, 7 edo 8 byte itzultzen dira, led-a itzalita edo piztuta dagoen esaten duen mezu bat da. Aztertu diren neurriak memoria (RAM eta ROM) eta CPU kontsumoa eta latentzia izan dira. ROM erabilerarako exekutagarrien eta liburutegien tamaina eduki dira kontutan, Californium-en kasuan izan ezik. Kasu honetan, Java inplementazio bat izanik, .jar fitxategia eduki da kontutan. RAM eta CPU kontsumoa Gnu/-Linux herramienta batekin neurtu da, *time*. Honek programa baten exekuzioari buruzko estatistikak erakusten ditu. Latentziarako, Round Trip Time (RTT)-a neurtu da. Eskakizuna bidali denetik, bueltan ACK jaso arteko denbora da RTT-a. Horretarako, *tcpdump* sare sniffer-a erabili da, sarean sortu diren pakete guztiak aztertzen dituena. 50 eskaera egin dira konbinazio guztiekin, segundu bat utziz eskaera bakoitzaren artean eta minimo, maximo, batazbestekoa eta mediana kalkulatu dira.

136	19.160598	192.168.1.105	192.168.1.103	CoAP	55	ACK, MID:37471, 2.05 Content (text/plain)
142	20.085344	192.168.1.103	192.168.1.105	CoAP	52	CON, MID:37472, PUT, /Led (text/plain)
147	20.098666	192.168.1.105	192.168.1.103	CoAP	56	ACK, MID:37472, 2.05 Content (text/plain)
150	21.085597	192.168.1.103	192.168.1.105	CoAP	52	CON, MID:37473, PUT, /Led (text/plain)
154	21.102892	192.168.1.105	192.168.1.103	CoAP	55	ACK, MID:37473, 2.05 Content (text/plain)
161	22.085833	192.168.1.103	192.168.1.105	CoAP	52	CON, MID:37474, PUT, /Led (text/plain)
168	22.105925	192.168.1.105	192.168.1.103	CoAP	56	ACK, MID:37474, 2.05 Content (text/plain)
172	23.086069	192.168.1.103	192.168.1.105	CoAP	52	CON, MID:37475, PUT, /Led (text/plain)
176	23.102247	192.168.1.105	192.168.1.103	CoAP	55	ACK, MID:37475, 2.05 Content (text/plain)
180	24.086282	192.168.1.103	192.168.1.105	CoAP	52	CON, MID:37476, PUT, /Led (text/plain)
184	24.098034	192.168.1.105	192.168.1.103	CoAP	56	ACK, MID:37476, 2.05 Content (text/plain)
188	25.086567	192.168.1.103	192.168.1.105	CoAP	52	CON, MID:37477, PUT, /Led (text/plain)
192	25.097275	192.168.1.105	192.168.1.103	CoAP	55	ACK, MID:37477, 2.05 Content (text/plain)
193	26.086832	192.168.1.103	192.168.1.105	CoAP	52	CON, MID:37478, PUT, /Led (text/plain)
201	26.092840	192.168.1.105	192.168.1.103	CoAP	56	ACK, MID:37478, 2.05 Content (text/plain)
204	27.087069	192.168.1.103	192.168.1.105	CoAP	52	CON, MID:37479, PUT, /Led (text/plain)
208	27.098280	192.168.1.105	192.168.1.103	CoAP	55	ACK, MID:37479, 2.05 Content (text/plain)
209	28.087300	192.168.1.103	192.168.1.105	CoAP	52	CON, MID:37480, PUT, /Led (text/plain)
213	28.098920	192.168.1.105	192.168.1.103	CoAP	56	ACK, MID:37480, 2.05 Content (text/plain)
217	29.087527	192.168.1.103	192.168.1.105	CoAP	52	CON, MID:37481, PUT, /Led (text/plain)
221	29.102699	192.168.1.105	192.168.1.103	CoAP	55	ACK, MID:37481, 2.05 Content (text/plain)
222	30.087818	192.168.1.103	192.168.1.105	CoAP	52	CON, MID:37482, PUT, /Led (text/plain)
226	30.098595	192.168.1.105	192.168.1.103	CoAP	56	ACK, MID:37482, 2.05 Content (text/plain)
227	31.088074	192.168.1.103	192.168.1.105	CoAP	52	CON, MID:37483, PUT, /Led (text/plain)
231	31.098638	192.168.1.105	192.168.1.103	CoAP	55	ACK, MID:37483, 2.05 Content (text/plain)
235	32.088324	192.168.1.103	192.168.1.105	CoAP	52	CON, MID:37484, PUT, /Led (text/plain)
242	32.111865	192.168.1.105	192.168.1.103	CoAP	56	ACK, MID:37484, 2.05 Content (text/plain)
245	33.088564	192.168.1.103	192.168.1.105	CoAP	52	CON, MID:37485, PUT, /Led (text/plain)
249	33.099199	192.168.1.105	192.168.1.103	CoAP	55	ACK, MID:37485, 2.05 Content (text/plain)
250	34.088769	192.168.1.103	192.168.1.105	CoAP	52	CON, MID:37486, PUT, /Led (text/plain)
254	34.102269	192.168.1.105	192.168.1.103	CoAP	56	ACK, MID:37486, 2.05 Content (text/plain)
255	35.089019	192.168.1.103	192.168.1.105	CoAP	52	CON, MID:37487, PUT, /Led (text/plain)
259	35.100373	192.168.1.105	192.168.1.103	CoAP	55	ACK, MID:37487, 2.05 Content (text/plain)
268	36.089251	192.168.1.103	192.168.1.105	CoAP	52	CON, MID:37488, PUT, /Led (text/plain)
272	36.111126	192.168.1.105	192.168.1.103	CoAP	56	ACK, MID:37488, 2.05 Content (text/plain)
277	37.089498	192.168.1.103	192.168.1.105	CoAP	52	CON, MID:37489, PUT, /Led (text/plain)

6.2 Irudia: Tcpdump-ek ateratako datuak

7. KAPITULUA

Emaitzak

Experimentua deskribatu ondoren, emaitzak erakusten dira kapitulu honetan. 4 kapituluan azaldu bezala, teorikoki CoAPy izan ezik beste liburutegi guztiek ondo funtzionatu beharko lukete haien artean. 7.1 taulan ageri dira interoperabilitatearen datuak, aurrikusitakoak dira. Horizontalean zerbitzariak ikusten dira eta bertikalean bezeroak. Test hau PUT eskaerak eginda eta led-a piztu edo itzali den ikusita egin da, ez da inongo sare analisirik egin.

	libcoap	smcp	microcoap	FreeCoAP	Californium	node-coap	CoAPthon	CoAPy
libcoap	+	+	+	+	+	+	+	–
smcp	+	+	+	+	+	+	+	–
FreeCoAP	+	+	+	+	+	+	+	–
Californium	+	+	+	+	+	+	+	–
h5.coap	+	+	+	+	+	+	+	–
node-coap	+	+	+	+	+	+	+	–
CoAPthon	+	+	+	+	+	+	+	–
CoAPy	–	–	–	–	–	–	–	+
Copper	+	+	+	+	+	+	+	–

7.1 Taula: Interoperabilitatea

Latentziari dagokionez, 7.2 taulak 50 eskaeren maximo (goian) eta minimoak (behean) erakusten ditu, konbinazio bakoitzerako. Neurriak milisegundutan daude eta dezimal batera borobildu dira. CoAPthon zerbitzaria eta h5.coap bezeroaren konbinazioak ez du guztiz ondo funtzionatzen. Birtransmisio asko gertatzen dira, CoAPthon-ek token bat sartzen duelako erantzunean, nahiz eta h5.coap-ek eskaeran sartu ez [Github, 2017]. Horrela, h5.coap-ek erantzuna jasotzean, token batekin jasotzen du, eta ez denez berak bidalitako eskaeraren berdina ACK baztertu eta birtransmisio bat egiten du, birtransmisio posible

maximora heldu arte. Hala ere, zerbitzariak led-a piztu edo itzali egiten du, arazoa bueltako ACK-n baitago eta ez eskaeran.

Orokorrean, ikus daiteke C inplementazioak azkarragoak direla zerbitzariaren aldean, FreeCoAP-en bezeroarekin izan ezik. Argiago ikusten da hau maximoetan, baina minimoetan ere ikus daiteke. Garrantzitsua da esatea Californium zerbitzariaren kasuan, lehenengo eskaera dela beti geldoena, tokenak kudeatu eta bezero bakoitza erregistratzen baitu, ondoren etorriko diren eskaerak azkarrago tratatzeko. Emaitza hauek logikoak dira, C lengoaia azkarragoa eta optimizatuagoa dela suposatzen delako. 4 kapituluan azaldu bezala, gainerako lengoaiek kapa gehigarri behar dute sistema eragilearen gainean.

	libcoap	smcp	microcoap	FreeCoAP	Californium	node-coap	CoAPthon	CoAPy
libcoap	16,1 4,0	40,8 2,6	223,4 6,7	33,5 3,0	75,6 6,0	109,6 7.0	104,3 10,4	- -
smcp	44,5 3,4	54,5 5,4	45,1 8,3	41,3 2,1	91,2 8,8	134,1 12,9	180,3 15,7	- -
FreeCoAP	226,3 6,8	124,1 5,4	551,6 8.0	37,6 3,2	87,1 5,1	100,8 12,6	113,8 12,4	- -
Californium	40,2 3,9	51,0 6,5	32,8 8,7	33,8 8,5	79.0 9,9	101,8 6,3	35,0 10,1	- -
h5.coap	24,3 3,1	12,7 5,6	14,0 4,5	18,1 5,6	94,4 7,6	107,5 7,6	- -	- -
node-coap	22,9 5,3	24,4 2,8	25,8 4,1	24,1 2,6	94,6 4.0	109,0 6.0	40,4 9,5	- -
CoAPthon	42,9 6,9	35,0 2,5	61,6 5,0	21,7 5,7	101.0 8.0	120.0 9,4	26,9 9,0	- -
CoAPy	- -	- -	- -	- -	- -	- -	- -	110,3 5,1

7.2 Taula: RTT Maximo eta minimoak, ms-tan

7.3 taulan 7.2 taulako eskaera berdinen batazbestekoa (goian) eta mediana (behean) erakusten dira. Milisegunduetan eta dezimal batera borobilduta, aurrekoan bezala. Datu hauek aztertzean, ondorioak berdinak dira, beraz ez du ematen zenbaki arraro batek datuak aldatu dituenik, koherenteak dira. Hala ere, espero ez den emaitza bat ageri da: h.coap, node-coap eta CoAPthon bezeroak azkarrak dira, ia C liburutegiak beste, nahiz eta Node.js eta Python inplementazioak izan.

	libcoap	smcp	microcoap	FreeCoAP	Californium	node-coap	CoAPthon	CoAPy
libcoap	5,5 5,0	9,5 5,3	22,2 13,9	12,5 10.0	17,3 11,4	19,8 13,9	23,2 19,4	- -
smcp	18,9 16,2	18,3 15,6	16,7 13,4	13,9 11,5	18,7 15,8	24,7 20,2	40,2 29,4	- -
FreeCoAP	30,6 21,1	27,7 18,7	48,8 21,6	15,9 13,4	20,7 19,2	24,5 21,7	30,0 27,0	- -
Californium	14,4 10,8	15,4 11,0	15,2 12,4	14,1 12,0	15,9 12,3	18,3 13,7	19,2 18,7	- -
h5.coap	9,6 9.0	8.0 8,1	8,3 8,1	8,2 8.0	13,3 10.0	17,7 12,7	- -	- -
node-coap	10,2 9,6	9,7 9,0	10,1 9,3	9,2 9.0	12,5 10,3	17.0 13,6	17,9 16,3	- -
CoAPthon	13,8 8,8	14,1 10,5	10,0 7,7	8,85 7,7	19,7 12,0	20,1 13,7	15,8 15,2	- -
CoAPy	- -	- -	- -	- -	- -	- -	- -	21.0 13,9

7.3 Taula: RTT-ren batazbestekoa eta mediana, ms-tan

Sistemaren baliabideen kontsumoari dagokionez, 7.4 taulak ROM erabilera erakusten du, bytetan. Bi C inplementazio (libcoap eta smcp) instalatu egin behar dira, beraz liburutegiaren lerroan instalatutako .a fitxategien tamaina erakusten da. FreeCoAP eta microcoap ez dira instalatu behar, CoAP dependentzia guztiak exekutagarriaren barnean daude. JavaScript (h5.node eta node-coap) eta Python (CoAPthon eta CoAPy) liburutegietan bakoitzaren *lib* karpetaren tamaina erakusten da liburutegiaren tamainan bezala. Java (Californium) liburutegiaren kasuan, beharrezko dependentzia guztiak .jar-aren barruan daude, beraz logikoa exekutagarria handiena izatea. Bezero eta zerbitzari lerroetan, bakoitzaren exekutagarriaren tamaina erakusten da, aurreko analisia egiteko sortutako bezero eta zerbitzariena hain zuzen ere. Neurriak aztertuta, argi dago, C inplementazioak astunaguak direla Python eta JavaScript-enak baino. Hala ere, gauza batzuk eduki behar dira kontutan: benchmarking honetarako sarrera/irteera liburutegiak ere erabili behar izan dira eta garrantzitsuagoa dena, lengoaia ez konpilagarriak erabilita kapa gehigarri bat behar da sistema eragilearen gainean. JavaScript inplementazioek Node.js exekuzio ingurunea behar dute, Python-ekoek interpretatzailea eta Javaren kasuan Java Virtual Machine-a, eta hauek tamainen arteko diferentzia baiko pisu handiagoa edukiko dute.

Azkenik, 7.5 taulak zerbitzari baten eta 1000 eskaera egiten dituen bezero baten exekuzioan kontsumitzen diren CPU eta RAM erakusten ditu. Neurri hauek Gnu/Linux he-

	libcoap	smcp	microcoap	FreeCoAP	Californium	h5.coap	node-coap	CoAPthon	CoAPy
Liburutegia	296150	383366	-	-	-	106097	47341	277095	76074
Zerbitzaria	21812	18356	18700	39772	4257012	-	1329	2508	4563
Bezeroa	33328	22684	-	31616	4257329	3621	883	1672	1794

7.4 Taula: ROM erabilera

rramienta bati, *time*, esker lortu dira, */usr/bin/time -v "zerbitzaria/bezeroa exekutatzeko komandoa"* exekutatuz terminalean. Komando honek exekuzioaren RAM memoria maximoa erakusten du, kbytetan, eta baita aplikazioa CPU-an egon den denbora ere, segunduetan, bai erabiltzaile eta baita sistemaren espazioan.

Neurriek erakusten dutenez, C inplementazioak dira arinenak eta azkarrenak. Aztertutako laurek 3.3Mbyte RAM inguru behar dute, Californium eta bi Node.js inplementazioek 10 aldiz astunagoak direlarik. Python inplementazioak ere nahiko arinak dira, baina C-koengandik urruti. CPU-aren aldetik, bereziki erabiltzaile espazioan, argi dago C inplementazioak azkarragoak direla, konpilazioa eta kode natiboa exekutatzeak ematen duen abantailari esker.

	libcoap	smcp	microcoap	FreeCoAP	Californium	h5.coap	node-coap	CoAPthon	CoAPy
Server User Time	0.09	0.04	0.13	0.06	2.30	-	2.35	6.60	0.99
Server System Time	0.13	0.08	0.09	0.13	0.27	-	0.24	1.23	0.16
Server Peak RAM	3252	3232	3236	3240	24492	-	31008	14348	8332
Client User Time	0.06	0.10	-	0.10	4.66	4.22	2.73	5.77	4.96
Client System Time	0.08	0.31	-	0.21	0.41	0.62	0.21	1.25	0.13
Client Peak RAM	3240	3312	-	3256	29676	37732	34484	12924	10644

7.5 Taula: CPU eta RAM erabilera

8. KAPITULUA

Ondorioak eta proiektuaren etorkizuna

Experimentuak burutu, neurriak hartu eta emaitzak aurkeztu ondoren, proiektuaren ondorioak eta beraren jarraipena azalduko dira. Egindako lana aztertzen da, ateratako ondorioak eta zer jarraipen posible izan dezakeen.

Alde batetik, proiektuari berari dagozkion helburuak kontutan hartuta, egiaztatu da liburutegi guztiak elkar interoperableak direla, CoAPy izan ezik. CoAPthon zerbitzaria eta h5.coap bezeroaren kasuan ere arazo bat dago, baina liburutegien inplementatzaileei idatzi zaie bug-a azalduz. Eraginkortasunaren aldetik, argi dago C inplementazioak direla arinenak. Kontutan edukita CoAP zerbitzariak izaten direla baliabideetan mugatuak diren gailuak (sentsoreak eta eragileak), zentzu asko dauka C erabiltzeak. C liburutegien artean, libcoap eta smcp dira azkarrenak, baina microcoap eta FreeCoAP magnitude orden bat gutxiago ROM behar dute inplementatzeko, beraz erabakia hardwarearen araberakoa izan beharko da. Gainera, kontutan eduki behar da microcoap-ek ez duela espezifikazio osoa inplementatzen eta horrek arazoak ekar ditzakela eta FreeCoAP-ek ez duela laguntza handirik ematen baliabideak sortu eta kudeatzeko. Bestalde, FreeCoAP GnuTLS liburutegiaren bidez DTLS erabiltzeko gai da, nahiz eta lan honetan gaitasun hau desaktibatuta egon den.

Bezeroen aldean berriz, disko beharra ez da hain garrantzitsua, orokorrean dispositibo handiagoak izango baitira. Horrela, Java, Node.js eta Python inplementazioek, harrigarria izan arren, libcoap eta smcp-ren abiaduratik ez dira urrun geratzen. Gainera, lengoaia hauek ematen duten abstrakzio mailari esker oso aukera honak izan daitezke teknologia horietan ibiltzen ohituta dauden garatzaileentzako. Horrez gain, bezeroa Cloud bat

baldin bada, oso asko erabiltzen dira lengoaia hauek bertan. Hala ere, CoAPy-k ez du gainontzekoekin funtzionatzen, zaharkitua geratu da eta proiektuak ez du aurrera jarraitzeko itxurarik, beraz, deskartatu egin daiteke zuzenean. Aipatzekoa da baita ere, Californium dela IETF-k aukeratutako inplementazioa beraien probetarako eta ETSI-ko Plugtest [ETSI, 2012] guztiak pasatzen dituela.

Eraginkortasunaz gain, erabiltzeko erraztasuna ere kontutan eduki behar da, hau da, garatzaile batek zenbat denbora beharko duen zerotik hasi eta zerbitzari bat martxan jartzen. Zentzu horretan libcoap, smcp, microcoap, Californium eta CoAPthon dira aukera onenak, baliabideen sorkuntza eta kudeaketa era garden batean egiten dutelako.

Beste aldetik, proiektuaren kudeaketari dagozkion ondorioak hauek dira: hasieratik proposatutako ataza guztiak bete dira. Liburutegiak aztertu dira, konparaketa teorikoa egin eta ondoren demo aplikazioak prestatu hainbat liburutegirekin. Demo horiekin probak egin dira eta neurriak hartu, analisi praktiko bat egiteko. Azkenik, artikulu zientifiko bat ere prestatu da eta konferentzia batera bidali da.

Bukatzeko, proiektuaren jarraipenerako hainbat bide posible detektatu dira. Alde batetik, gailu mugatuagoetan antzeko analisi bat egitea. Gailu merkeago eta txikiagoak elkar konektatzea bada helburua baliabideetan mugatuagoak egongo dira. Adibidez ARM Cortex M4 plataforma oso erabilia da industrian. Bestalde, eskalabilitatea aztertzea ere garrantzitsua izan daiteke, Python, Java edo Node.js liburutegiek abantaila izan baitezakete Cloud-aren bizkarrezurrean, konexio asko ondo kudeatzeko gai baitira teknologia hauek. Azkenik, MQTT, AMQP eta DDS bezalako beste protokolo arin batzuek gailu mugatuetan konparatzea ere interesgarria da.

Eranskinak

A. ERANSKINA

Artikulua

Ondorengo orrialdeetan nazioarteko konferentzia batera bidalitako artikulua dago, bertara bidalitako formatu berarekin.

Available online at www.sciencedirect.com

SciVerse ScienceDirect

Procedia Computer Science 00 (2016) 000–000

Procedia Computer Science

www.elsevier.com/locate/procedia

The 8th International Conference on Ambient Systems, Networks and Technologies (ANT 2017)

Theoretical and Empirical Analysis of CoAP Implementations for Industrial Internet of Things: A Survey

Markel Iglesias-Urkia, Adrián Orive, Aitor Urbieta

IK4-Ikerlan Technology Research Centre, Information and Communication Technologies Area.
Pº J.M.Arizmendiarrieta, 2. 20500 Arrasate-Mondragón, Spain

Abstract

Over the last few years, the Internet of Things (IoT) has grown in protocols, implementations and use cases. In terms of communication protocols, the Constrained Application Protocol (CoAP) stands out among the rest. This is extremely lightweight and capable of running in resource constrained devices and networks. There exist many implementations of CoAP, each of these with its own particular features and requirements. Therefore, it is important to choose the CoAP implementation that suits better to the specific requirements of each application. This paper presents a theoretical analysis and an empirical comparison of several open source CoAP implementations. First of all, it surveys current CoAP implementations, and compares them in terms of built-in core, extensions, target platform, programming language and interoperability. Then, it analyzes their performance in terms of latency, memory and CPU consumption in a real testbed deployed in an industrial scenario.

Keywords: Benchmarking, CoAP, IIoT, Internet of Things, IoT, Industrial Internet of Things, Lightweight, Protocols, Survey

1. Introduction

"The Internet of Things is a system of interrelated computing devices, mechanical and digital machines, objects, animals or people that are provided with unique identifiers and the ability to transfer data over a network without requiring human-to-human or human-to-computer interaction"[1]. According to Evans[2] and Chase[3], by 2020, 50 billion devices will be connected to the IoT while Jeon[4] further claims that they will reach 75 billion. Even though these predictions are probably too optimistic (as by 2015 we are far behind the expected growth[2]), the amount of interconnected objects grows daily at a fast pace thanks to the communication protocols.

* Corresponding author. Tel.: +34 943 712 400 ; fax: +34 943 796 944.
** ORCID: http://orcid.org/{0000-0001-7708-3252, 0000-0003-2919-5799, 0000-0001-5836-4198}
E-mail address: {miglesias,aorive,aurbieta}@ikerlan.es

Some common Internet protocols, such as HTTP[5], have been originally used to connect the first IoT devices but new, lighter ones have emerged, to fit the constrained requirements imposed by IoT environments, including CoAP[6], MQTT[7], AMQP[8] and DDS[9] among others. Even though it is one of the newest protocols, CoAP is getting some momentum as it follows the REST paradigm, making the adaptation process from HTTP easy for developers. Besides, it is very light both in terms of device and network requirements.

One of the final goals of the IoT industry is to get to as many different kind of devices as possible. As said in the previous paragraph, CoAP could be a good fit to do so, but it is very important to correctly choose among the wide pool of existing implementations. The selected one has to support the requirements of the system and it has to correctly balance the required CPU and memory resources and energy consumption.

The main objective of the work presented in this paper is to ease the selection of the CoAP implementation that better fits each project by analyzing different open source implementations both theoretically and empirically. The selection of the proper implementation for each project must be based both on offered features and performance. The implementations selected for the analysis target several platforms and are written in different languages. All the tests have been conducted on an industrial deployment over the Raspberry Pi platform, analyzing the response time or latency, as well as the memory and CPU needed to deploy the implementations.

The rest of this paper is organized as follows. First, related work is presented. Next, CoAP is described along with some extensions and implementations. After that, in Section 4, a theoretical comparison of different implementations is carried out. Following section describes the experiment set-up and the measured parameters. The results are presented in Section 6 and finally conclusions and guidance for future work are offered in Section 7.

2. Related Work

There is some previous work analyzing CoAP's performance in different platforms and its comparison against other IoT protocols. Early on, when CoAP's standardization was not complete, there was some work made analyzing different implementations, like Lerche et al.[10] and Villaverde et al.[11] In the latter, the authors analyze CoAP theoretically short after the first draft of CoAP was presented and analyze some of the implementations from that time. The paper presents a list of available implementations and some conclusions based on other papers' analysis. Lerche et al.[10] summarize the results of the first ETSI CoAP Plugtest[12]. They present the participant implementations and their interoperability but they do not offer a performance analysis of any sort. Both Lerche et al.[10] and Villaverde et al.[11] present a list of available implementations, but since they were written, CoAP's standardization went on and new implementations have shown up, so they are not up to date. To research more current work on CoAP's performance, it is necessary to widen the search parameters to other protocols or focus on a sigle implementation on different hardware.

Focusing on CoAP, Ludovici et al.[13] present their own implementation for TinyOS called TinyCoAP. They compare its performance against HTTP/TCP, HTTP/UDP and the original TinyOS CoAP implementation, CoAPBlip, on TelosB motes and they measure latencies and memory and energy consumption. Kruger et al.[14] present a benchmarking of the same CoAP implementation on different hardware, i.e. Raspberry Pi, BeagleBone and BeagleBone Black. They compare the performance on class 4 and class 10 SD cards and with running, off and uninstalled GUI. The parameters they measure are latency, bandwidth and CPU and memory usage, all in the loopback interface. They also compare the latency on a TelosB mote server with a BeagleBone gateway and a laptop client.

Widening the analyzed protocols scope, CoAP and HTTP have similar structure even though they target different environments. Colitti et al.[15] work on Tmote Sky and Zolertia Z1 motes to measure response time and energy consumption. Kuladinithi et al.[16] present their own implementation called libcoap and port it to Contiki and TinyOS to compare the performance against HTTP. Elmangoush et al.[17] present CoAP, HTTP, MQTT and AMQP, but they only work on the two formers. They use the OpenMTC platform to measure the bandwidth per request interval time, the response time per request interval time and the response time for different payload sizes.

Another popular IoT protocol is MQTT and De Caro et al.[18] measure several parameters such as the latency, the bandwidth usage and the package loss ratio over different QoS and network loss configurations on Smartphone implementations of CoAP and MQTT. Thangavel et al.[19] present a common middleware based for both MQTT and CoAP and measure the latency and bandwidth consumption.

There are more popular IoT protocols besides CoAP and MQTT. Talaminos-Barroso et al.[20] implement a tool for an eHealth application with the possibility of using DDS, MQTT, CoAP, JMS, AMQP and XMPP. They benchmark

the CPU and memory usage, the bandwidth consumption and the messages' latency and jitter. Mun et al.[21] selected CoAP, MQTT, MQTT-SN, WebSockets and TCP. The authors aim to ease programmers make a good choice selecting the protocol that better fits for their applications, and to do so, they measure the performance, the energy efficiency and the memory and CPU usage. Chen et al.[22] analyze MQTT, CoAP, DDS and a custom protocol over UDP. They use a network emulator to configure different parameters and measure consumed bandwidth, latency and package loss.

As it has been presented in previous paragraphs, CoAP's performance has been compared against other protocols, such as HTTP, MQTT and others. Some CoAP implementations' performance has also been studied on different hardware. However, the only work done analyzing different implementations of CoAP is based on early drafts of the CoAP specification, therefore it is not up to date. Besides, it analyzes their interoperability and main features, it does not conduct a performance analysis. Thus, this paper's goal is to fill that gap offering an up to date comparison, both empirical and theoretical, to help system designers choose the implementation that better fits their requirements.

3. CoAP

Published as RFC 7252[23] by the IETF CoRE Working Group[24] on June 2014, the Constrained Application Protocol[6] is a web transfer protocol designed for resource constrained devices and networks. It is built on top of UDP and it follows the REST paradigm, so it works similar to HTTP. It uses a subset of HTTP's request verbs: GET, POST, PUT and DELETE; also response codes and content-formats, even though in this last case there in an additional one. The default port for CoAP is 5683 and the one for secure CoAP 5684. A CoAP URI consists on the following format: *coap://host[:port][/path][?query]*. The user needs to know the path in order to access the available resources, and it will also be able to send queries to the server in the URI. As it is implemented over UDP, CoAP does not guarantee the arrival of the packets, and therefore implements two main types of messages that differ in whether they require acknowledgment: Confirmable and Non-confirmable. The other two types are Reset and ACK, which can piggyback data. To securize the connections, CoAP can not use TLS due to the underlying UDP protocol as it requires ordered and guaranteed message delivery. DTLS is UDP's alternative to TLS, but it also pays the cost of losing some of the benefits of using UDP derived from not requiring an open connection.

The IETF proposes some extensions to broaden the capabilities of the CoAP specifications:

- **Constrained RESTful Environments (CoRE) Link Format**[25]: this extension defines the format for the links that constrained servers use to describe their resources, attributes and relationships between links.
- **Block-Wise Transfers in the Constrained Application Protocol (CoAP)**[26]: this extension allows large payloads to be sent when needed (e.g. firmware updates), avoiding IP fragmentation.
- **CoRE Resource Directory**[27]: the IETF proposes an entity called Resource Directory, which has the information about resources of other CoAP servers, allowing battery saving and making the server discovery easy.
- **Observing Resources in the Constrained Application Protocol (CoAP)**[28]: this extension enables CoAP servers to send push notifications to clients, like other publish/subscribe protocols such as MQTT or XMPP.
- **Group Communication for the Constrained Application Protocol (CoAP)**[29]: this extension explains how CoAP should be used in a multicast environment.
- **CoAP Simple Congestion Control/Advanced CoCoA**[30]: the CoAP specification proposes basic behaviors to avoid network congestion but to add more sophisticated methods, the IETF is working on this draft.

There are many available CoAP implementations with different features and targets. The aim of this work is to use and compare open source libraries that target several platforms and environments. We tried to cover different programming languages and runtime environments and to do so the following implementations have been selected:

- **libcoap**[31] is a library written in C and it is designed to fit in a wide range of devices, from embedded devices to big POSIX ones. It supports the official RFC 7252 for the client and server side and it also provides support for several extensions, i.e. Observe mode, Block-Wise transfer and Resource Directory. The source code comes with very complete examples and it is designed to easily add DTLS with OpenSSL or tinydtls.
- **smcp**[32] is another C library. It aims to be implemented in devices from bare-metal sensors to Linux-based devices, including embedded ones. It supports client and server sides following the RFC 7252 and it is possible

to use it with BSD sockets or μIP. As for CoAP extensions, it supports the Observe mode and Multicast groups. It provides a command line client called smcpctl and has an, at this time, experimental DTLS branch.

- **microcoap**[33] is a limited CoAP library in C that targets small microcontrollers. The source code provides examples for POSIX and Arduino. It follows the RFC 7252 but it does not support it entirely. It supports only the server side and has limited features. It does not support DELETE requests, only GET, PUT and POST, the ACKs can only be piggybacked and it does not support retries.
- **FreeCoAP**[34] is the last C library analyzed in this paper. It targets GNU/Linux devices as it uses GnuTLS for security even if it has a new tinydtls branch. It follows the stable specification RFC 7252.
- **Californium**[35] is a very complete Java library for not so constrained devices. It targets backends with JVM and it offers both client and server sides. In addition to the RFC 7252, it also supports some extensions, i.e. Observe mode, Block-Wise transfer and Resource Directory. It has optional DTLS support with the Scandium project.
- **h5.coap**[36] is a JavaScript library that targets the Node.js platform. It provides only the client side and it follows the stable definition of the RFC 7252. In addition, it supports the Observe mode and the Block-Wise transfer.
- **node-coap**[37] is another JavaScript library for Node.js. It provides both client and server sides and follows the stable version of the RFC. It supports not only the core but also the Observe mode and Block-Wise transfer extension.
- **CoAPthon**[38] is a RFC 7252 compliant Python library that supports both client and server sides. In addition to the core features, it also supports Observe mode, Core-Link format, Multicast and Block-Wise transfer extensions.
- **CoAPy**[39] is another Python library. It follows an old draft of CoAP (draft-ietf-core-coap-02) making it theoretically not compatible with the others. In addition to CoAP's specification, it also supports Block-Wise transfer.

Despite the list above that covers different targets and languages, there are other implementations worth mentioning that have been discarded for different reasons. Copper[40] is a visual client implemented as a Firefox plugin. Erbium[41] is a widely used C implementation, targeted towards ContikiOS and TinyCoAP[42] targets tinyOS. As in this paper the working environment is going to be Raspberry Pi, they do not fit.

4. Theoretical and Feature Comparison

Once the different implementations overview has been presented, the next step is to start comparing them in order to offer a guideline for the library selection from a theoretical point of view. To get this, the first step is to compare the libraries based on their characteristics (language, target, extensions, etc.) and then to describe how the library integration process is for new implementations.

Table 1 summarizes the features of the analyzed libraries. In the first place the used version of each library is listed. The first differentiating characteristic is the programming language in which they are written and the targeted platform, being four of them written in C and one in Java, while the four left are evenly distributed between Python and Node.js (Javascript). All of them are supposed to comply with RFC 7252 except for CoAPy, that is based on a previous draft (draft-02). Regarding client and server implementations, they all support both sides except for microcoap (server side only) and h5.coap (client side only). Most of them also implement the most important extensions: the Observe mode and the Block-Wise transfer. In addition, the most mature ones (i.e. libcoap and Californium) also support the Resource Discovery extension.

When considering interoperability, CoAPy uses some options such as deprecated Path-Uri, where code 9 is used. The next draft (draft-03) changed this option to code 11, thus making implementations from the previous drafts non-interoperable. The rest of the libraries claim to use the latest specification and are therefore expected to be interoperable.

Regarding performance, the first four libraries are expected to be faster and more lightweight. This is due to the fact that they are developed in native C instead of a non-native language that requires an additional layer: Java Virtual Machine (JVM) for Californium, Node.js and the V8 JavaScript engine for JavaScript libraries and the Python interpreter for Python ones.

Table 1. Characteristics of the implementations

Library	Version	Language	Target platform	Specification	Client/Server	Extensions*	Notes
libcoap	Develop Sept. 24, 2016	C	POSIX, Contiki, lwIP, TinyOS	RFC 7252	Client & Server	Observe, Block-Wise, Resource directory	–
smcp	Master Sept. 24, 2016	C	Embedded devices, bare-metal sensors, Linux-based devices	RFC 7252	Client & Server	Observe, Multicast	–
microcoap	Master Sept. 24, 2016	C	Arduino, POSIX	RFC 7252	Server	–	Partial Support
FreeCoAP	Master Sept. 24, 2016	C	GNU/Linux	RFC 7252	Client & Server	–	–
Californium	1.1.0-SNAPSHOT	Java	JVM supporting devices	RFC 7252	Client & Server	Observe, Blockwise Resource Directory	–
h5.coap	0.0.0	JavaScript	Node.js supporting devices	RFC 7252	Client	Observe, Block-Wise	–
node-coap	0.18.0	JavaScript	Node.js supporting devices	RFC 7252	Client & Server	Observe, Block-Wise	–
CoAPthon	Master Sept. 24, 2016	Python	Python supporting devices	RFC 7252	Client & Server	Observe, Core-Link Multicast, Block-Wise	–
CoAPy	0.0.3-DEV	Python	Python supporting device	Draft-2	Client & Server	Blockwise	Inactive

* Even if most of them do not explicitly say it, they support Core Link-Format extension.

Some of the implementations are more mature than others, this has made that few of them have evolved not only to offer basic functionalities but also to provide advanced mechanisms to deal with resources, available request types and response codes. Due to this, the creation and management of resources varies between implementations and this affects considerably the development process of applications, as it is described in the following lines:

- **libcoap**: it has an interface which makes adding new resources very easy. The library itself manages the response codes, so the developers only need to add the name of the resource, which request types it supports and link each kind of request to a handler.
- **smcp**: similar to libcoap, the developers only need to create the handlers and resources and add them to the system with the help of an interface. The library handles the response codes.
- **microcoap**: to add new resources to the server, they have to be defined and added to a resource array along with the handlers. The library manages everything else by itself.
- **FreeCoAP**: the response codes have to be defined by the developers and the resources' path too, but this implementation does not include an interface to ease the handling of resources. Adding new resources and handling the response codes or actions is a bit tricky.
- **Californium**: this is a very mature implementation and it makes it easy to add and manage resources. To add a new resource, a Java class needs to be created, with the handlers for the different types of supported requests. Then, through an interface, the resources are easily added to the server and the library itself handles the rest.
- **node-coap**: the handling of resources and response codes is on the developers' hands. This implementation does not provide an interface to ease the creation of resources and management of the response codes, so the handling of resources and request has to be made by the application itself, not the library.
- **CoAPthon**: a Python class has to be created for each resource, with the methods that the application support. The library manages by itself the response codes and everything else.
- **CoAPy**: a Python class has to be created for each resource, but the application needs to handle the response codes, the library does not provide this feature.

To summarize, libcoap, smcp, microcoap, Californium and CoAPthon are the easiest libraries to build a server with, because they all handle the response codes within the library itself. The developers just need to define the resources and the handlers and link then to the server, the library handles everything else. The rest of libraries require the developer to handle this explicitly adding unneeded complexity to the application.

5. Experiment Setup

Current industry solutions mainly use wired networks such as Profibus, Modbus, CAN, Profinet, etc. These are not very flexible and modifying their set-ups requires wiring changes; so as wireless networks are becoming more reliable, their acceptance as a valid alternative is growing. The present experiment has been deployed on an industrial prototype

over Raspberry Pi-s. This platform is widely used in industrial prototyping and gateways, having the advantage of being able to run a wider pool of implementations than more constrained platforms. In this case, as both client and server were needed, two Raspberry Pi 3 model B were connected via WiFi through a local 56 Mbps router.

The listed libraries have been tested unmodified (except for FreeCoAP, which has been ported to IPv4) both in terms of interoperability and performance. All implementations have been tested against each other except h5.coap server microcoap client as they are not implemented. The requests have a single byte payload, while the responses' ones are alternatively 7 and 8 bytes long. However, it is important to note that at the time that the tests have been conducted it has been discovered that CoAPthon does not allow to add any payload to 2.04 Changed responses[43], thus it sends less bytes in the response, saving time and resources. Regarding the metrics, memory usage, CPU consumption and latency have been measured. For ROM usage, both the executable and library files' sizes have been taken into account. RAM and CPU consumption has been analyzed with the GNU/Linux time tool, which shows statistics of an execution. Round Trip Time (RTT) has been selected to measure latency with *Tcpdump* network sniffer on the client. 50 requests have been sent for each combination of client and server, with a single second waiting interval between send requests.

6. Results

Once the experiment scenario has been described, the results are in order. As explained in Section 4, all implementations are theoretically interoperable with the exception of CoAPy. Table 2 shows the results of the interoperability test between different implementations, which are the same as expected. This test has been performed sending PUT requests to the server and observing its response, there has been no network analysis or sniffing. During the performance test an interoperability issue between h5.coap client and CoAPthon server was found that was not obvious on the previous test. CoAP messages use two kind of identifiers, message identifiers that allow to pair a message with its acknowledge and tokens for a more generic purpouse, that may be empty. CoAPthon server is generating a token when the client sends an empty one contrary to RFC 7252, and h5.coap is checking the tokens to match in order to accept the acknowledgement, so they are being rejected and the message is sent again until the maximum allowed number of retries.

Table 2. Implementation interoperability

Client \ Server	libcoap	smcp	microcoap	FreeCoAP	Californium	node-coap	CoAPthon	CoAPy
libcoap	✔	✔	✔	✔	✔	✔	✔	–
smcp	✔	✔	✔	✔	✔	✔	✔	–
FreeCoAP	✔	✔	✔	✔	✔	✔	✔	–
Californium	✔	✔	✔	✔	✔	✔	✔	–
h5.coap	✔	✔	✔	✔	✔	✔	✔	–
node-coap	✔	✔	✔	✔	✔	✔	✔	–
CoAPthon	✔	✔	✔	✔	✔	✔	✔	–
CoAPy	–	–	–	–	–	–	–	✔

Regarding the latency, Table 3 shows the median (left column) and maximum (right) values in milliseconds (rounded to the first decimal), to represent normal and worst case scenarios for every server-client combination. As expected, libcoap, smcp and microcoap are faster servers, as C is a lower level language not requiring additional abstraction layers and thus being more optimized. Java (Californium), Node.js (h5.coap and node-coap) and Python (CoAPthon) implementations have given surprisingly good results as clients even in comparison to most of the C ones, being libcoap (C) the fastest library for most cases.

Table 3. Median and Max of RTT in ms

Client \ Server	libcoap		smcp		microcoap		FreeCoAP		Californium		node-coap		CoAPthon		CoAPy	
libcoap	5,0	16,1	5,3	40,8	13,9	223,4	10.0	33,5	11,4	75,6	13,9	109,6	19,4	104,3	–	–
smcp	16,2	44,5	15,6	54,5	13,4	45,1	11,5	41,3	15,8	91,2	20,2	134,1	29,4	180,3	–	–
FreeCoAP	21,1	226,3	18,7	124,1	21,6	551,6	13,4	37,6	19,2	87,1	21,7	100,8	27,0	113,8	–	–
Californium	10,8	40,2	11,0	51,0	12,4	32,8	12,0	33,8	12,3	79.0	13,7	101,8	18,7	35,0	–	–
h5.coap	9.0	24,3	8,1	12,7	8,1	14,0	8.0	18,1	10.0	94,4	12,7	107,5	–	–	–	–
node-coap	9,6	22,9	9,0	24,4	9,3	25,8	9.0	24,1	10,3	94,6	13,6	109,0	16,3	40,4	–	–
CoAPthon	8,8	42,9	10,5	35,0	7,7	61,6	7,7	21,7	12,0	101.0	13,7	120.0	15,2	26,9	–	–
CoAPy	–	–	–	–	–	–	–	–	–	–	–	–	–	–	13,9	110,3

Considering the system resource consumption, Table 4 shows the ROM usage in bytes. Some of the studied libraries (libcoap, smcp, h5.coap, node-coap, CoAPthon and CoAPy) need to be installed, showing either the *.a* files' or *lib* folder sizes in the library row; while the rest (microcoap, FreeCoAP and Californium) include all the dependencies in the executable. In the Server and Client rows, the sizes of the respective executables are shown, corresponding to the examples that have been used for the previous analysis. C implementations have bigger executable and library sources than Python or JavaScript, while Java's executables are considerably heavier. The size of I/O libraries has not been taken into account except for Californium and using non native languages require adding a runtime environment (Node.js), an interpreter (Python) or a Virtual Machine (Java), which is heavier than the size difference.

Table 4. ROM usage

	libcoap	smcp	microcoap	FreeCoAP	Californium	h5.coap	node-coap	CoAPthon	CoAPy
Library	296150	383366	-	-	-	106097	47341	277095	76074
Server	21812	18356	18700	39772	4257012	-	1329	2508	4563
Client	33328	22684	-	31616	4257329	3621	883	1672	1794

Table 5 shows the CPU and RAM usage of a server (left column) and a client (right) execution for 1000 requests. The data has been collected using the Gnu/Linux tool *time*, executing */usr/bin/time -v "server/client execution command"*. This command shows the peak usage of the RAM (KB) and the total CPU time (seconds) the application has used, both in user and system space. The results show that in execution time C implementations are the fastest and most lightweight. All four tested C implementations are similar in terms of RAM consumption with about 3.3 MB, while Californium and both Node.js implementations are around 10 times heavier. Python implementations are more lightweight but are still far behind C ones. In terms of CPU usage, especially the User Time, it is also in clear favour of C implementations, which are much faster due to compilation and execution of native code.

Table 5. CPU and RAM usage

	libcoap		smcp		microcoap		FreeCoAP		Californium		h5.coap		node-coap		CoAPthon		CoAPy	
User Time	0.09	0.06	0.04	0.10	0.13	-	0.06	0.10	2.30	4.66	-	4.22	2.35	2.73	6.60	5.77	0.99	4.96
System Time	0.13	0.08	0.08	0.31	0.09	-	0.13	0.21	0.27	0.41	-	0.62	0.24	0.21	1.23	1.25	0.16	0.13
Peak RAM	3252	3240	3232	3312	3236	-	3240	3256	24492	29676	-	37732	31008	34484	14348	12924	8332	10644

7. Conclusion

In this paper we have theoretically and experimentally compared several CoAP implementations. From this comparison, we confirm that libcoap, smcp, microcoap, FreeCoAP, Californium, node-coap and CoAPthon are interoperable, while CoAPy is not, mainly because it is based on an outdated draft. Regarding server performance, C-based implementations stand out. Among them, libcoap and smcp are the fastest libraries, while microcoap's and FreeCoAP's memory requirements are one order of magnitude lower. However, microcoap does to not include all the specifications of the RFC 7252 and FreeCoAP does not handle response code generation tasks and URIs in a transparent way. At the client side, where disk space requirements are not critical, the Java, Node.js and Python implementations are surprisingly close to libcoap and smcp in terms of speed. Moreover, thanks to the higher abstraction level of their languages, Californium (Java), CoAPthon (Python), h5.coap and node-coap (Node.js) are recommended for CoAP clients.

From this work, different lines of development arise. On one hand, the evaluation of the proposed libraries in highly constrained platforms, such as devices based on the ARM Cortex-M architecture, would be valuable. On the other hand, a comparison of CoAP libraries in terms of scalability may also be relevant, since the libraries based on Java, Python or Node.js may overcome the performance of C-based libraries in larger scenarios. Finally, a comparison between implementations of CoAP and other IoT lightweight protocols, such as MQTT, AMQP and DDS, may also provide useful insights for their use in resource constrained platforms.

Acknowledgment

This work is partially supported by the Basque Government's Elkartek program within the LANA II project (Grant agreement no. KK-2016/00052).

References

1. Internet of Things (IoT). http://internetofthingsagenda.techtarget.com/definition/Internet-of-Things-IoT
2. D. Evans. The Internet of Things, How the Next Evolution of the Internet Is Changing Everything. *Cisco White Paper*, 2011.
3. J. Chase. The Evolution of the Internet of Things. *Texas Instruments White Paper*, 2013.
4. J. Jeon. Web Browser as Universal client for IoT. http://www.slideshare.net/hollobit/web-browser-as-universal-client-for-iot, 2011.
5. HTTP. https://www.ietf.org/rfc/rfc2616.txt
6. CoAP. http://coap.technology/
7. MQTT. http://mqtt.org/
8. AMQP. https://www.amqp.org/
9. DDS. http://portals.omg.org/dds/
10. C. Lerche and K. Hartke and M. Kovatsch. Industry adoption of the Internet of Things: A constrained application protocol survey. *IEEE International Conference on Emerging Technologies and Factory Automation, ETFA*, 2012.
11. B.C. Villaverde and D. Pesch and R. De Paz Alberola and S. Fedor and M. Boubekeur. Constrained Application Protocol for Low Power Embedded Networks: A Survey. *Sixth International Conference on Innovative Mobile and Internet Services in Ubiquitous Computing*, 2012
12. IoT CoAP Plugtests http://www.etsi.org/plugtests/coap/coap.htm
13. A. Ludovici and P. Moreno and A. Calveras. TinyCoAP: A Novel Constrained Application Protocol (CoAP) Implementation for Embedding RESTful Web Services in Wireless Sensor Networks Based on TinyOS. *Journal of Sensor and Actuator Networks*, 2013.
14. C.P. Kruger and G.P. Hancke. Benchmarking Internet of things devices. Proceedings. *12th IEEE International Conference on Industrial Informatics, INDIN 2014*, 611-616, 2014.
15. W. Colitti and K. Steenhaut and N. De Caro and V. Buta and V. Dobrota. Evaluation of constrained application protocol for wireless sensor networks. *18th IEEE Workshop on18th IEEE Workshop on Local & Metropolitan Area Networks (LANMAN)*, 1-6, 2011.
16. K. Kuladinithi and O. Bergmann and T. Pötsch and M. Becker and C. Görg. Implementation of coap and its application in transport logistics. *Proc. IP+SN*, 2011.
17. A. Elmangoush and R. Steinke and T. Magedanz and A.A. Corici and A. Bourreau and A. Al-Hezmi. Application-derived communication protocol selection in M2M platforms for smart cities. *International Conference on Intelligence in Next Generation Networks, ICIN*, 2015.
18. N. De Caro and W. Colitti and K. Steenhaut and G. Mangino and G. Reali. Comparison of two lightweight protocols for smartphone-based sensing. *IEEE SCVT 2013 - Proceedings of 20th IEEE Symposium on Communications and Vehicular Technology in the BeNeLux*, 0-5, 2013.
19. D. Thangavel and X. Ma and A. Valera and H.X. Tan and C.K. Tan. Performance evaluation of MQTT and CoAP via a common middleware. *IEEE ISSNIP 2014 - 2014 IEEE 9th International Conference on Intelligent Sensors, Sensor Networks and Information Processing, Conference Proceedings*, 21-24, 2014.
20. A. Talaminos-Barroso and M.A. Estudillo-Valderrama and L.M. Roa and J. Reina-Tosina and F. Ortega-Ruiz. A Machine-to-Machine protocol benchmark for eHealth applications - Use case: Respiratory rehabilitation. *Computer Methods and Programs in Biomedicine*, 1-11, 2016.
21. D.-h .Mun and M.L. Dinh and Y.-w. Kwon. An Assessment of Internet of Things Protocols for Resource-Constrained Applications. *IEEE 40th Annual Computer Software and Applications Conference*, 2016.
22. Y. Chen and T. Kunz. Performance evaluation of IoT Protocols under a Constrained Wireless Access Network. *International Conference on Selected Topics in Mobile and Wireless Networking, MoWNeT*, 2016.
23. RFC 7252. https://tools.ietf.org/html/rfc7252
24. IETF CoRE Working Group. https://datatracker.ietf.org/wg/core/charter/
25. Constrained RESTful Environments (CoRE) Link Format. https://tools.ietf.org/html/rfc6690
26. Block-Wise Transfers in the Constrained Application Protocol (CoAP). https://tools.ietf.org/html/rfc7959
27. CoRE Resource Directory draft-ietf-core-resource-directory-08. https://tools.ietf.org/html/draft-ietf-core-resource-directory-08
28. Observing Resources in the Constrained Application Protocol (CoAP). https://tools.ietf.org/html/rfc7641
29. Group Communication for the Constrained Application Protocol (CoAP). https://tools.ietf.org/html/rfc7390
30. CoAP Simple Congestion Control/Advanced draft-bormann-core-cocoa-04. https://tools.ietf.org/html/draft-bormann-core-cocoa-04
31. libcoap. https://libcoap.net/
32. smcp. https://github.com/darconeous/smcp
33. microcoap. https://github.com/1248/microcoap
34. FreeCoAP. https://github.com/keith-cullen/FreeCoAP
35. Californium. https://eclipse.org/californium/
36. h5.coap. https://github.com/morkai/h5.coap
37. node-coap. https://github.com/mcollina/node-coap
38. CoAPthon. https://github.com/Tanganelli/CoAPthon
39. CoAPy. http://coapy.sourceforge.net/
40. Copper. https://github.com/mkovatsc/Copper
41. Erbium. http://people.inf.ethz.ch/mkovatsc/erbium.php
42. TinyCoAP. https://github.com/AleLudovici/TinyCoAP
43. CoAPthon PUT response. https://github.com/Tanganelli/CoAPthon/issues/45

Bibliografia

[1248, 2016] 1248 (2016). microcoap. https://github.com/1248/microcoap.

[AleLudovici, 2013] AleLudovici (2013). TinyCoAP. https://github.com/AleLudovici/TinyCoAP.

[Bergmann, 2016] Bergmann, O. (2016). libcoap: C-Implementation of CoAP. https://libcoap.net/.

[Bormann et al., 2015] Bormann, C., Betzler, A., Gomez, C., and Demirkol, I. (2015). CoAP Simple Congestion Control/Advanced. https://tools.ietf.org/html/draft-bormann-core-cocoa-03.

[Bormann and Shelby, 2016] Bormann, C. and Shelby, Z. (2016). Block-wise transfers in CoAP. https://tools.ietf.org/html/draft-ietf-core-block-20.

[Chase, 2013] Chase, J. (2013). The evolution of the internet of things. Texas Instruments.

[Chen and Kunz, 2016] Chen, Y. and Kunz, T. (2016). Performance evaluation of IoT protocols under a constrained wireless access network. *2016 International Conference on Selected Topics in Mobile and Wireless Networking, MoWNeT 2016*.

[CoAPy, 2010] CoAPy (2010). CoAPy. http://coapy.sourceforge.net/.

[Colitti et al., 2011] Colitti, W., Steenhaut, K., De Caro, N., Buta, B., and Dobrota, V. (2011). Evaluation of constrained application protocol for wireless sensor networks. *18th IEEE Workshop on18th IEEE Workshop on Local & Metropolitan Area Networks (LANMAN), 2011*.

[Cullen, 2016] Cullen, K. (2016). FreeCoAP. https://github.com/keith-cullen/FreeCoAP.

[De Caro et al., 2013] De Caro, N., Colitti, W., Steenhaut, K., Mangino, G., and Reali, G. (2013). Comparison of two lightweight protocols for smartphone-based sensing. *IEEE SCVT 2013 - Proceedings of 20th IEEE Symposium on Communications and Vehicular Technology in the BeNeLux*.

[Elmangoush et al., 2015] Elmangoush, A., Steinke, R., Magedanz, T., Corici, A. A., Bourreau, A., and Al-Hezmi, A. (2015). Application-derived communication protocol selection in M2M platforms for smart cities. In *2015 18th International Conference on Intelligence in Next Generation Networks, ICIN 2015*.

[ETH Zurich, 2014] ETH Zurich (2014). Erbium. http://people.inf.ethz.ch/mkovatsc/erbium.php.

[ETSI, 2012] ETSI (2012). IoT CoAP Plugtests. http://www.etsi.org/plugtests/coap/coap.htm.

[Evans, 2011] Evans, D. (2011). The internet of things. how the next evolution of the internet is changing everything. Cisco.

[Github, 2016] Github (2016). PUT response with payload. https://github.com/Tanganelli/CoAPthon/issues/45.

[Github, 2017] Github (2017). Server auto-token generation brakes RFC 7252. https://github.com/Tanganelli/CoAPthon/issues/48.

[Hartke, 2015] Hartke, K. (2015). Observing Resources in the Constrained Application Protocol (CoAP). https://tools.ietf.org/html/rfc7641.

[Kovatsch, 2014] Kovatsch, M. (2014). Copper (Cu) CoAP user-agent for Firefox. http://people.inf.ethz.ch/mkovatsc/copper.php.

[Kruger and Hancke, 2014] Kruger, C. P. and Hancke, G. P. (2014). Benchmarking Internet of things devices. *Proceedings - 2014 12th IEEE International Conference on Industrial Informatics, INDIN 2014*.

[Kuladinithi et al., 2011] Kuladinithi, K., Bergmann, O., Thomas Pötsch, Becker, M., and Görg, C. (2011). Implementation of coap and its application in transport logistics. *Proc. IP+ SN, ...*

[Lerche et al., 2012] Lerche, C., Hartke, K., and Kovatsch, M. (2012). Industry adoption of the Internet of Things: A constrained application protocol survey. *IEEE International Conference on Emerging Technologies and Factory Automation, ETFA*.

[Ludovici et al., 2013] Ludovici, A., Moreno, P., and Calveras, A. (2013). TinyCoAP: A Novel Constrained Application Protocol (CoAP) Implementation for Embedding RESTful Web Services in Wireless Sensor Networks Based on TinyOS. *Journal of Sensor and Actuator Networks*.

[mccollina, 2016] mccollina (2016). node-coap. https://github.com/mcollina/node-coap.

[morkai, 2014] morkai (2014). h5.coap. https://github.com/morkai/h5.coap.

[Mun et al., 2016] Mun, D.-h., Dinh, M. L., and Kwon, Y.-w. (2016). An Assessment of Internet of Things Protocols for Resource-Constrained Applications. In *Computer Software and Applications Conference (COMPSAC), 2016 IEEE 40th Annual*.

[Quattlebaum, 2016] Quattlebaum, R. (2016). SMCP - A Full-Featured Emedded CoAP Stack. https://github.com/darconeous/smcp.

[Rahman and Dijk, 2014] Rahman, A. and Dijk, E. (2014). Group Communication for the Constrained Application Protocol (CoAP). https://tools.ietf.org/html/rfc7390.

[Shelby, 2012] Shelby, Z. (2012). Constrained RESTful Environments (CoRE) Link Format. https://tools.ietf.org/html/rfc6690.

[Shelby et al., 2014] Shelby, Z., Hartke, K., and Bormann, C. (2014). The Constrained Application Protocol (CoAP). https://tools.ietf.org/html/rfc7252.

[Shelby et al., 2016] Shelby, Z., Koster, M., Bormann, C., and van der Stok, P. (2016). CoRE Resource Directory. https://tools.ietf.org/html/draft-ietf-core-resource-directory-07.

[Talaminos-Barroso et al., 2016] Talaminos-Barroso, A., Estudillo-Valderrama, M. A., Roa, L. M., Reina-Tosina, J., and Ortega-Ruiz, F. (2016). A Machine-to-Machine protocol benchmark for eHealth applications - Use case: Respiratory rehabilitation. *Computer Methods and Programs in Biomedicine*.

[Tanganelli, 2016] Tanganelli (2016). CoAPthon. https://github.com/Tanganelli/CoAPthon.

[Thangavel et al., 2014] Thangavel, D., Ma, X., Valera, A., Tan, H. X., and Tan, C. K. Y. (2014). Performance evaluation of MQTT and CoAP via a common middleware. *IEEE*

ISSNIP 2014 - 2014 IEEE 9th International Conference on Intelligent Sensors, Sensor Networks and Information Processing, Conference Proceedings.

[The Eclipse Foundation, 2014] The Eclipse Foundation (2014). Californium. http://www.eclipse.org/californium/.

[Villaverde et al., 2012] Villaverde, B. C., Pesch, D., De Paz Alberola, R., Fedor, S., and Boubekeur, M. (2012). Constrained application protocol for low power embedded networks: A survey. In *Proceedings - 6th International Conference on Innovative Mobile and Internet Services in Ubiquitous Computing, IMIS 2012*.