Aplicaţie de tip browser FS (Server/Client CoAP)

Mihalache Nicoleta-Ecaterina

Grupa 1309A

Petrov Sergiu

Grupa 1309B

1) Descrierea temei

CoAP - Constrained Application Protocol este un protocol specializat de aplicaţii Internet pentru dispozitive constrânse, aşa cum este definit în RFC 7252. Acesta permite dispozitivelor să comunice prin Internet. Protocolul este destinat în special pentru hardware-ul restricţionat, cum ar fi microcontrolere pe 8 biţi, senzori de putere redusă şi dispozitive similare care nu pot rula pe HTTP sau TLS. Este o simplificare a protocolului HTTP care rulează pe UDP, care ajută la economisirea lăţimii de bandă. Este conceput pentru a fi utilizat între dispozitive din aceeaşi reţea constrânsă (de exemplu, reţele cu consum redus de energie, cu pierderi), între dispozitive şi noduri generale de pe Internet şi între dispozitive din diferite reţele constrânse, ambele conectate prin internet. CoAP este, de asemenea, utilizat prin alte mecanisme, cum ar fi SMS-urile pe reţelele de comunicaţii mobile.

Ce este un **nod de reţea**? Un nod este orice dispozitiv fizic dintr-o reţea de alte dispozitive care poate trimite, primi şi da mai departe informaţii. Computerul este cel mai des întâlnit nod şi se numeşte adesea nodul computerului sau nodul de internet.Un nod de reţea este de obicei orice dispozitiv care primeşte şi apoi comunica ceva prin reţea, dar poate primi şi salva doar datele, transmiţând informaţii în alta parte sau poate crea şi trimite date.

Ce este **UDP**? User Datagram Protocol(UDP) este un protocol de comunicatie pentru calculatoare care împreuna cu Internet Protocol (IP), face posibila livrarea mesajelor într-o reţea. Este similar cu sistemul postal, în sensul ca pachetele de informaţii (corespondenţa) sunt trimise în general fără confirmare de primire, în speranta ca ele vor ajunge, fara a exista o legatura efectiva între expeditor si destinatar.

Principalele caracteristici ale acestui protocol sunt:

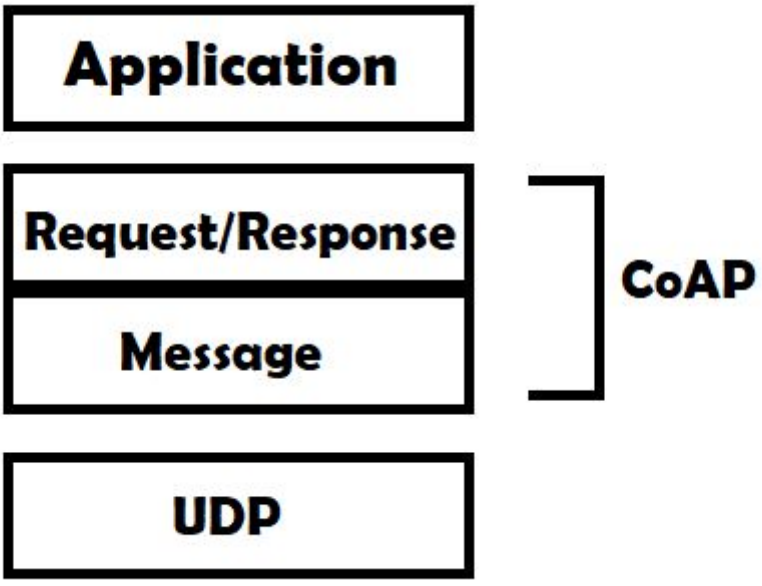
- Protocol web utilizat în M2M cu cerinţe constrânse;
- Schimb de mesaje asincron;
- Foarte simplu de analizat;

- URI şi suport de tip conţinut;
- Capabilităţi proxy şi cache;

Unele dintre cazurile specifice în care CoAP sunt utile sunt:

Hardware-ul dvs. nu poate rula HTTP sau TLS: dacă acesta este cazul, atunci rularea CoAP şi DTLS poate face practic acelaşi lucru ca HTTP. Dacă cineva este expert în API-uri HTTP, atunci migrarea va fi simplă. Primeşti GET pentru citire şi POST, PUT şi DELETE pentru mutaţii şi securitatea rulează pe DTLS.

Hardware-ul dvs. foloseşte bateria: dacă aceasta este o problemă, atunci rularea CoAP va îmbunătăţi performanţa bateriei în comparaţie cu HTTP prin TCP / IP. UDP economiseşte o anumită lăţime de bandă şi face protocolul mai eficient.



2)Detalii implementare

- Cele două echipe trebuie să colaboreze în vederea implementării unei soluţii de accesare a unui sistem de fişiere (FS) la distanţă;
- Server-ul va pune la dispoziţie resursele FS
- Clientul va fi capabil să execute toată gama de operaţii standard (acces, creare, modificare, ştergere) pentru foldere şi fişiere

-Operaţii standard:

1)Acces -metoda **GET**

2)Creare -metoda **PUT**

3)Modificare -metoda **POST** şi metoda **PUT**

4)Ştergere -metoda **DELETE**

- Cele două aplicaţii trebuie să suporte un număr de coduri (vezi formatul mesajului) care să includă - codul 0.00 (mesaj fără conţinut), metodele GET, POST (vezi Method Codes) şi o metodă nouă propusă de echipe în contextul temei (fiţi inventivi!), codurile de răspuns relevante pentru aplicaţie:

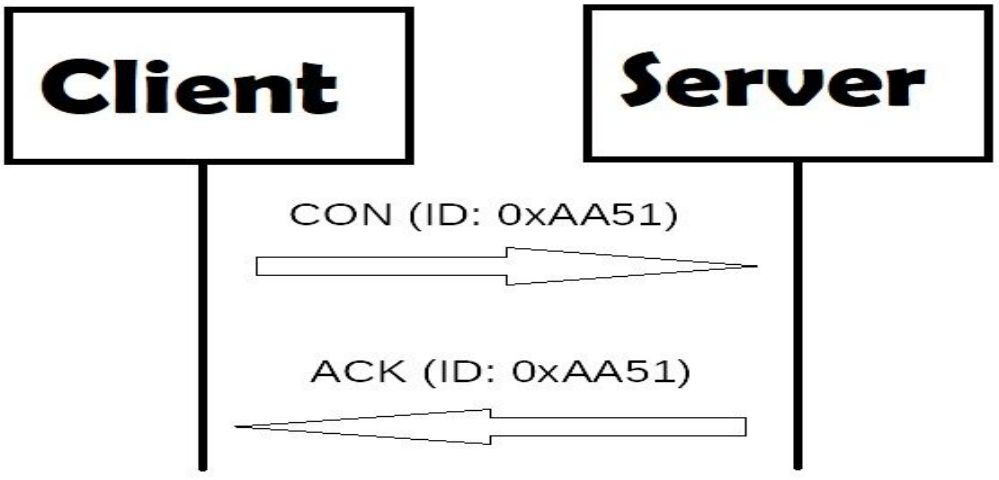
Mesajul CoAP este cel mai mic strat şi se ocupă cu schimbul de mesaje UDP între puncte finale. Mesajul CoAP are un ID unic şi trei părţi:

- un antet binar (32 byte)
- o serie de opţiuni compacte
- sarcină utilă

Protocolul CoAP utilizează două tipuri de mesaje:

- Mesaj confirmabil
- Mesaj neconfirmabil

Un mesaj confirmabil CoAP este un mesaj de încredere. În timpul schimbului de mesaje între două puncte finale, aceste mesaje pot fi fiabile. În protocolul CoAP, un mesaj de încredere este obţinut folosind un mesaj confirmabil (CON). Folosind acest tip de mesaj, clientul poate fi sigur că mesajul va ajunge la server. Un mesaj de confirmare CoAP este trimis din nou şi din nou până când cealaltă parte trimite un mesaj de confirmare (ACK). Mesajul ACK conţine acelaşi ID al mesajului confirmabil (CON).

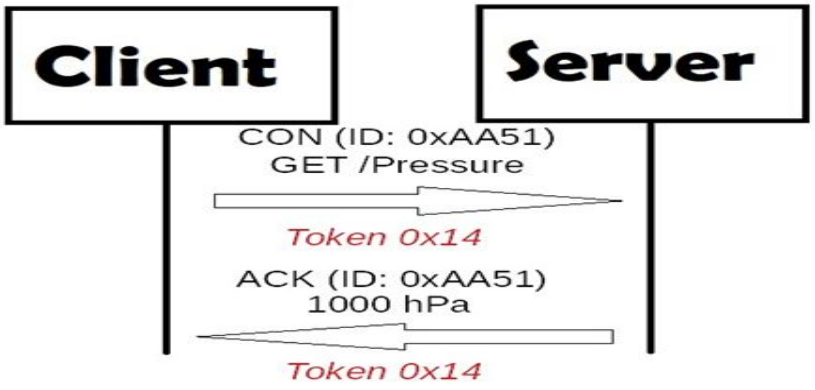


În diagrama de mai sus, putem vedea comunicarea, dar dacă serverul are probleme la gestionarea cererii primite, acesta poate trimite înapoi un mesaj Rest (RST) în loc de mesajul Confirm (ACK).

Mesaje neconfirmabile (NON) care nu necesită o confirmare de către server. Aceste mesaje sunt mesaje nesigure, nu conţin informaţii critice care trebuie livrate serverului. La această categorie aparţin mesaje care conţin valori citite de senzori. Chiar dacă aceste mesaje nu sunt fiabile, au un ID unic.

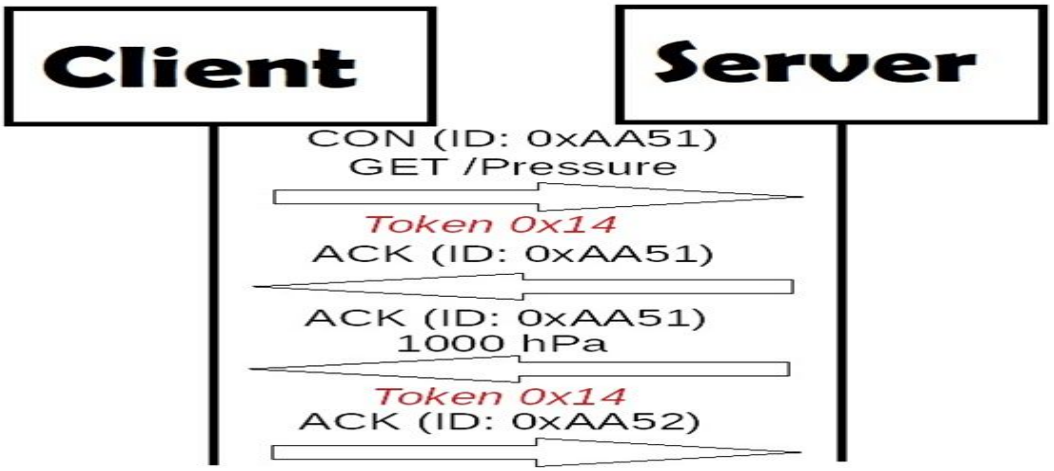
Acesta este al doilea strat din stratul de abstractizare. Aici cererea este trimisă utilizând un mesaj Confirmable (CON) sau Non-Confirmable (NON). Există mai multe scenarii în funcţie de dacă serverul poate răspunde imediat la solicitarea clientului sau răspunsul dacă nu este disponibil:

Dacă serverul poate răspunde imediat la solicitarea clientului, atunci dacă cererea este efectuată utilizând un mesaj confirmabil (CON), atunci serverul trimite înapoi clientului un mesaj de confirmare care conţine răspunsul sau codul de eroare:



Aici Tokenul este diferit de ID-ul mesajului şi este utilizat pentru a se potrivi cu cererea şi răspunsul.

Dacă serverul nu poate răspunde la cerere, atunci serverul trimite o confirmare cu un răspuns gol. De îndată ce răspunsul este disponibil, serverul trimite un nou mesaj confirmabil către client care conţine răspunsul. În acest moment, clientul trimite înapoi un mesaj de confirmare:



Dacă solicitarea care vine de la client este efectuată utilizând un mesaj care nu poate fi confirmat, atunci serverul răspunde utilizând un mesaj care nu poate fi confirmat.

Formatul mesajului:

Ver	T	TKL	Code	Message ID
Token				
Options (if exists..)				
Payload (if exists..)				

Unde:

- **Ver**: Este un număr întreg nesemnat de 2 biţi care indică versiunea
- **T**: este un număr întreg de 2 biți nesemnat care indică tipul mesajului: 0 confirmabil, 1 neconfirmabil;
- **TKL**: Lungimea jetonului este lungimea jetonului de 4 biţi
- **Cod**: este răspunsul la cod (lungimea de 8 biţi)
- **ID mesaj**: este ID-ul mesajului exprimat cu 16 biţi;
- **Token:**
- **Options:**
- **Payload**:

1)Metoda **GET** preia o reprezentare pentru informaţiile care în prezent

corespund resursei identificate prin URI-ul de solicitare. Dacă solicitarea include o opţiune de acceptare, aceasta indică formatul de conţinut preferat al unui răspuns. Dacă cererea include Opţiunea ETag, metoda GET solicită validarea ETag şi reprezentarea să fie transferată numai dacă validarea a eşuat. Pentru succes în răspuns va fi un cod de răspuns 2.05 (conţinut) sau 2.03 (valid). Metoda GET este sigură şi idempotentă.

2)Metoda **PUT** solicită ca resursa identificată prin cerere

URI-ul să fie actualizat sau creat cu reprezentarea anexată. Formatul de reprezentare este specificat de tipul de conţinut şi conţinut, codarea dată în opţiunea format-conţinut, dacă este furnizată. Dacă există o resursă la cererea URI, reprezentarea anexată trebuie să fie considerată o versiune modificată a resursei respective şi o versiune 2.04(Modificat) codul de răspuns trebuie returnat. Dacă nu există resursă serverul poate crea o nouă resursă cu acel URI, rezultând un cod de răspuns 2.01 (creat). Dacă resursa nu a putut fi creată sau modificata ar trebui să fie trimis un cod de răspuns de eroare adecvat. PUT nu este sigur, dar este idempotent.

3)Metoda **POST** solicită ca reprezentarea inclusă în

cerere să fie procesată. Funcţia efectivă a metodei POST este determinată de serverul de origine şi depinde de resursă ţintă. De obicei, rezultă crearea unei noi resurse sau actualizarea resursei ţintă. Dacă a fost creată o resursă pe server, răspunsul returnat de către server trebuie să aibă un cod de răspuns 2.01 (creat). Dacă POST reuşeşte dar nu are ca rezultat crearea unei noi resurse serverul trimite un cod de răspuns 2.04 (modificat). POST nu este nici sigur, nici idempotent.

4)Metoda **DELETE** realizeaza ştergerea unei resurse.Un cod de răspuns 2.02 (şters)reprezinta succesul operatiei sau în cazul în care resursa nu a existat înainte de cerere. DELETE nu este sigur, dar este idempotent.

O metodă HTTP este idempotentă dacă se poate face o cerere identică odată sau de mai multe ori la rând cu acelaşi efect, lăsând serverul în aceeaşi stare. Cu alte cuvinte, o metodă idempotentă nu ar trebui să aibă efecte secundare .

- Aplicaţiile trebuie să suporte mecanismul de comunicaţie cu confirmare, cât şi mesaje fără confirmare (selectabil din GUI)

-pe GUI vom crea un checkbox pentru mecanismul de confirmare/neconfirmare a mesajului;

-Pentru conectarea UDP-ului la stiva de comunicatii vom folosi biblioteca din Python: Socket, iar pentru realizarea interfeţei grafice vom folosi Tkinter.

3)Modalitatea de lucru propusă

Identificarea şi alocarea task-urilor

Task ID	Descriere task	Membru echipa
task 1	Documentaţie tema	m1,m2
task 2	Descriere protocol CoAp	m1
task 3	Scheletul aplicaţiei	m2
task 4	Implementarea interfeţei	m1, m2
task 5	Implementare metode(method codes)+cod	m1, m2
task 6	Testare aplicaţie	m1, m2
task 7	Raport final	m1, m2

Referinţe:

https://tools.ietf.org/html/rfc7252

https://medium.com/@harshhvm/what-is-coap-protocol-coap-protocol-introduction-overview-3e8bac4d7f8e

https://tools.ietf.org/id/draft-ietf-core-coap-04.xml