Aplicație de tip browser FS (Server CoAP)

Lupusoru Alexandru
Mîrț Alexandru

Grupa 1305B, Facultatea de Automatică și Calculatoare,
Universitatea “Gh. Asachi” Iași

Cuprins

1. Introducere

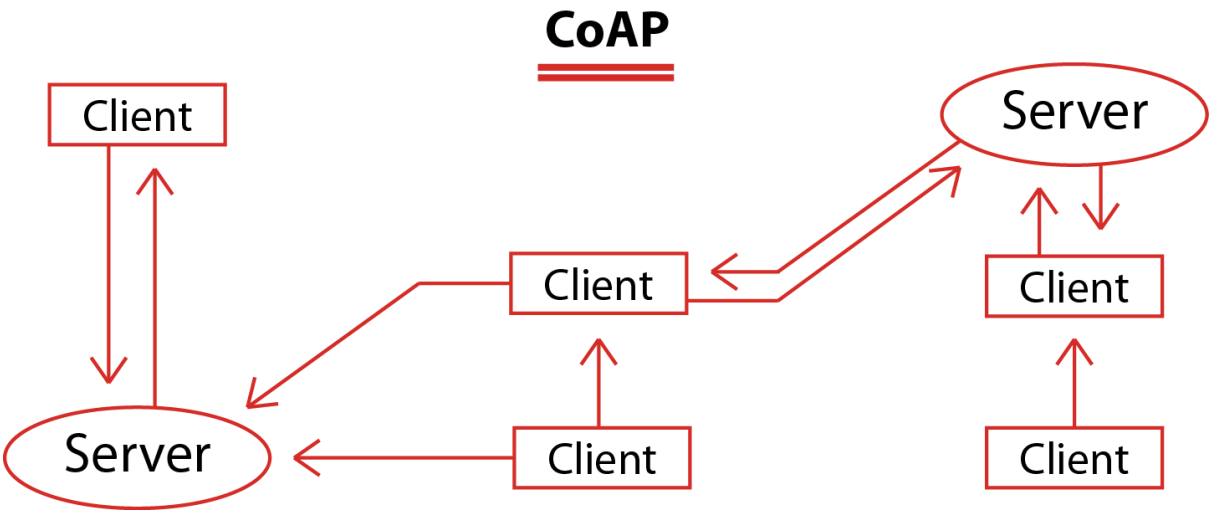
1.1 Protocolul de aplicație constrâns (CoAP)

Protocolul de aplicație constrâns (CoAP) este un protocol specializat de transfer pentru utilizare cu noduri și rețele constrânse (de exemplu, rețele cu consum redus de energie, cu pierderi). Nodurile au adesea microcontrolere pe 8 biti cu cantități mici de ROM și RAM, în timp ce sunt rețele restricționate cum ar fi IPv6 prin rețele de zonă personală fără fir de consum redus (6LoWPAN) au adesea rate de eroare ridicate ale pachetelor și un randament tipic de 10s de kbit/s. Protocolul este conceput pentru aplicații de tipul masina la mașină (M2M), cum ar fi energia inteligentă și automatizarea.

CoAP oferă un model de interacțiune cerere/răspuns între punctele finale ale aplicației, acceptă descoperirea serviciilor și resurselor și include concepte cheie ale web-ului, cum ar fi URI-uri și Internet Media. CoAP este conceput pentru a interfața cu ușurință cu HTTP pentru integrarea cu Web în timp ce îndeplinește cerințe specializate cum ar fi suport multicast, cheltuieli reduse și simplitate pentru medii constrânse.

Modelul de interacțiune al CoAP este similar cu cel al modelului client/server al HTTP-ului. Totuși interacțiunile M2M în general sunt într-o implementare CoAP atât pentru client, cât și pentru server.O cerere CoAP este echivalentă cu cea HTTP și este trimisă de client pentru a cere o acțiune pe o resursa (identificată prin URI) pe un server. Apoi serverul trimite un răspuns cu un cod de răspuns.

Comparativ cu HTTP, CoAP rezolvă schimbul asincron printr-un transport al datagramelor utilizator precum UDP.

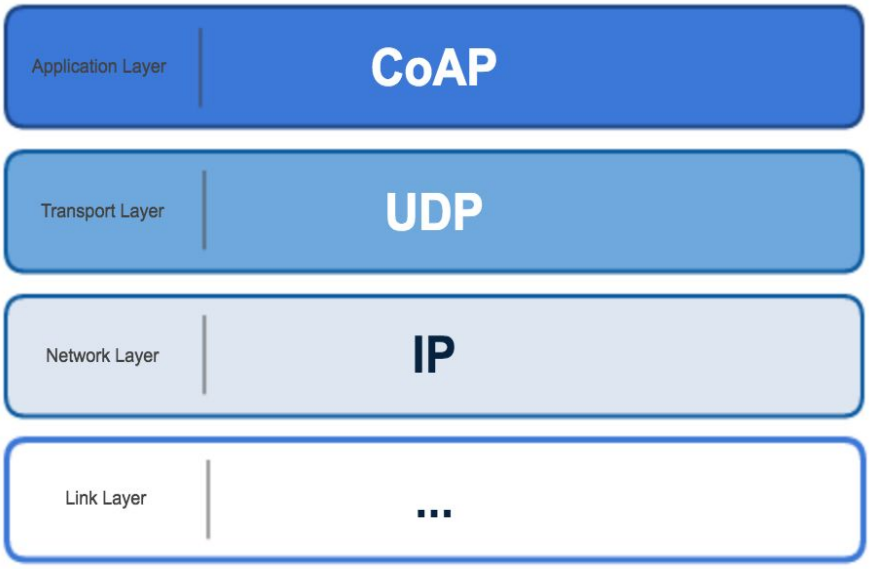


Specificația de baza a acestuia recomanda utilizarea standard a nivelului de transport prin mesaje de tip datagrama (UDP). Mecanismele responsabile pentru transmiterea și recepționarea în ordine a pachetelor, precum și pentru tratarea erorilor de rețea sunt implementate la nivelul CoAP prin intermediul mesajelor ”confirmabile”.

1.2 User Datagram Protocol (UDP)

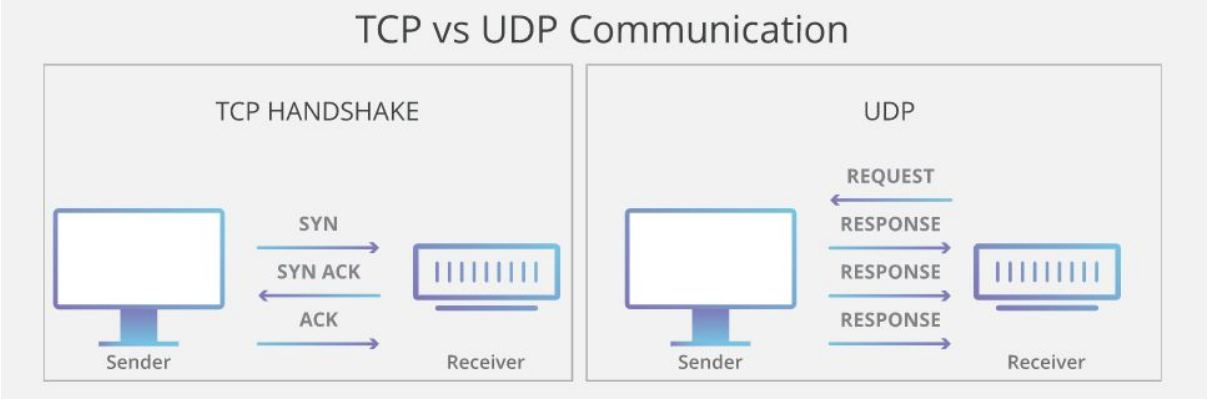
User Datagram Protocol (UDP), este un protocol de comunicație pentru calculatoare ce aparține nivelului Transport (nivelul 4) al modelului standard OSI.

Împreună cu Internet Protocol (IP), acesta face posibilă livrarea mesajelor într-o rețea. Spre deosebire de protocolul TCP, UDP constituie modul de comunicație fără conexiune. Este similar cu sistemul poștal, în sensul că pachetele de informații (corespondența) sunt trimise în general fără confirmare de primire, în speranța că ele vor ajunge, fără a exista o legătură efectivă între expeditor și destinatar. Practic, UDP este un protocol ce nu oferă siguranța sosirii datelor la destinație (nu dispune de mecanisme de confirmare); totodată nu dispune nici de mecanisme de verificare a ordinii de sosire a datagramelor sau a datagramelor duplicate. UDP dispune, totuși, în formatul datagramelor, de sume de control pentru verificarea integrității datelor sau de informații privind numărul portului pentru adresarea diferitelor funcții la sursa/destinație.



Caracteristicile de baza ale UDP îl fac util pentru diferite aplicații:

- orientat către tranzacții - util în aplicații simple de tip întrebare-răspuns cum ar fi DNS
- este simplu foarte util în aplicații de configurări, precum DHCP sau TFTP (Trivial FTP)
- lipsa întârzierilor de retransmisie îl pretează pentru aplicații în timp real ca VoIP, jocuri online
- lucrează excelent în medii de comunicații unidirecționale precum furnizarea de informații broadcast, în servicii de descoperire (discovery services), sau în partajarea de informații către alte noduri (RIP)



CoAP este bazat pe schimbul de mesaje prin UDP între punctele finale.

2. Implementare

2.1. Cerințe

- Cele două echipe (server și client) trebuie să colaboreze în vederea implementării unei soluții de accesare a unui sistem de fișiere (FS) la distanță
- Server-ul va pune la dispoziție resursele FS
- Clientul va fi capabil să execute toată gama de operații standard (acces, creare, modificare, ștergere) pentru foldere și fișiere
- Cele două aplicații trebuie să suporte un număr de coduri (vezi formatul mesajului) care să includă - codul 0.00 (mesaj fără conținut), metodele GET, POST (vezi Method Codes) și o metodă nouă propusă de echipe în contextul temei (fiți inventivi!), codurile de răspuns relevante pentru aplicație
- Aplicațiile trebuie să suporte mecanismul de comunicație cu confirmare, cât și mesaje fără confirmare (selectabil din GUI)

2.2.1 Mesajele

Eficienta CoAP deriva din codificarea binară a mesajelor și utilizarea unui antet fix de numai 4 octeți. Deși nu exista nicio restricție în ceea ce privește protocolul de nivel transport folosit, standardul recomandă folosirea UDP si impune o modalitate de transfer fara fragmentare. De aici rezultă că dimensiunea maximă a unui mesaj CoAP poate fi de 1152 de octeți, din care 1024 reprezintă incarcatura propriu-zisa. Antetul mesajului (figura 2.1) conține o serie de informat, ii referitoare la tipul mesajului transmis, identificatorul acestuia (relevant mai ales în cazul transferurilor de date cu confirmare din partea receptorului), precum și un jeton (”token”) care permite distinct, ia între cerere și răspuns.

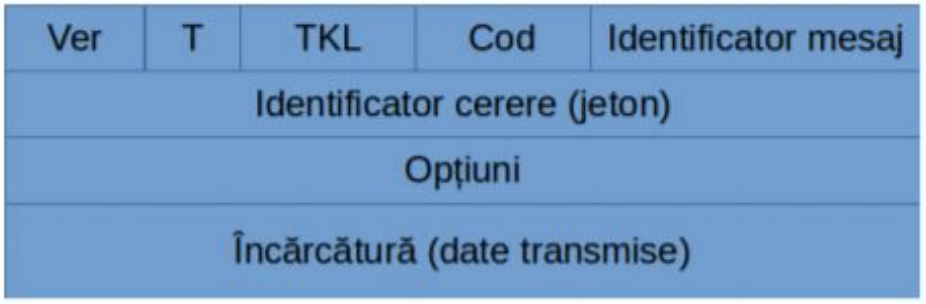


Figura 2.1: Structura unui mesaj CoAP

2.2.2 Adresarea datelor

Similaritatea cu HTTP se manifesta și prin modul în care datele sunt identificate. Mai concret, se utilizeaz˘a adresarea prin intermediul unui identificator unic de resurse (URI) cu distinct, ia că domeniul de adresare (namespace) folosit este fie coap, fie coaps pentru comunicarea securizată.

Formatul URI este următorul coap(s)://domeniu:port/resursă?filtru:

domeniul - reprezintă, în mod similar cu HTTP, un identificator al gazdei care cont, ine resursele (serverul CoAP)

port - poate fi opțional (se considera implicit porturile 5683 și 5684)

resursă - identificatorul unic al resursei care poate fi citită, scrisă s.a.m.d.

filtru - pentru interogarea prin intermediul unui filtru a unei resurse

2.2.3 Modelul de interacțiune

Modelul de interacțiune este bazat pe binomul cerere-răspuns și definește cele patru verbe de baza (codificate cu ajutorul campului cod din antetul pachetului) ale protocolului : Observăm deci că nu exista nicio restricție în ceea ce privește utilizarea principiilor arhitecturale REST. Corespunzător, o serie de răspunsuri grupate după tip (succes, eroare de transmisie și eroare de procesare) sunt definite de către standard. Toate aceste informații sunt codificate cu ajutorul unui singur octet care este împărțit după cum urmează :

3 biti - reprezinta clasa tipului de răspuns (succes, eroare de transmisie/client, eroare de procesare/server)

5 biti - reprezinta raspunsul asemanator cu cel din HTTP (nu există chiar o asociere 1:1)

2.2.4 Comenzi de executat

Metodă	Descriere
GET	Citirea datelor de la o resursă
POST	Cerere pentru procesarea conținutului din mesaj (depinde de implementarea resursei)
PUT	Modificarea unei resurse deja existente sau crearea alteia noi cu informațiile din conținut
DELETE	Ștergerea unei resurse

Lista de metode disponibile in CoAP

Serverul nostru implementează functionalitate, precum și răspuns pentru următoarele opțiuni de comenzi care pot veni din partea clientului:

- **back**, codificată prin '\x01', care ne întoarce în directorul părinte al fișierului în care ne aflăm la un moment dat. Nu se poate apela din directorul rădăcină *root* și returnează nouă cale spre directorul nou deschis.
- **open**, codificata prin '\x02' și urmată de numele fișierului sau directorului din cadrul folderului în care ne aflăm momentan pe care dorim să-l deschidem. În cazul acesteia vom trimite raspuns înapoi spre server calea spre noul fișier/director deschis.
- **save,** codificata prin '\x03' și urmată de numele fișierului, iar apoi de conținutul pe care dorim să-l salvam în fișier. Aceasta comanda poate fi apelată doar după ce am deschis un fișier și nu returneaza nimic în incarcatura mesajului raspuns.
- **new file**, codificata prin '\x04' și urmată de numele fișierului pe care dorim sa il cream. Aceasta trebuie apelată neapărat din interiorul directorului în care vrem să creăm nou fișier și nu returneaza nimic în incarcatura mesajului raspuns.
- **new directory**, codificata prin '\x05' și urmată de numele directorului pe care dorim sa il cream. Aceasta trebuie apelată neapărat din interiorul directorului în care vrem să creăm nou folder si nu returneaza nimic în incarcatura mesajului raspuns.
- **delete**, codificata prin '\x06' și urmată de numele fișierului sau directorului pe care dorim sa il stergem. Aceasta trebuie apelată neapărat din interiorul folderului părinte și șterge fișierul identificat, precum și copiii acestuia în cazul în care este de tipul director.

3. Clase

3.1. Interface

Clasa care răspunde de interfața grafica pe care o folosim în proiect, realizata cu biblioteca tkinter. Este vorba despre o singura fereastra suficient de simplă, care ne permite prin intermediul a doua butoane sa pornim și să oprim serverul. În continuare, sunt afișate conexiunile cu clientul (actualizate la primirea unui mesaj), iar mai jos acțiunile realizate de server ca raspuns la cererile primite de la client. În partea dreapta, este actualizată constant ierarhia de fișiere cu care lucram.

3.2. CoAP

Reține informațiile standard ale protocolului CoAP, precum tipurile și clasele mesajelor, de exemplu. Totodată, conține și o metoda statică care primește un obiect de tipul CoAPMessage și îl transforma in bytes, cum va trebui sa fie transmis către client.

```
@staticmethod
def wrap(msg: CoAPMessage) -> bytes:
    coap_header = CoAP.build_header(msg)
    payload = b''
    if len(msg.payload):
        payload = CoAP.PAYLOAD_MARKER + msg.payload.encode('utf-8')
    return coap_header + payload

@staticmethod
def build_header(msg: CoAPMessage) -> bytes:
    header = 0x00
    header |= msg.header_version << CoAP.VERSION
    header |= (0b11 & msg.msg_type) << CoAP.MSG_TYPE
    header |= msg.token_length << CoAP.TOKEN_LENGTH
    header |= (0b111 & msg.msg_class) << CoAP.MSG_CLASS
    header |= (0x1F & msg.msg_code) << CoAP.MSG_CODE
    header |= (0xFFFF & msg.msg_id) << CoAP.MSG_ID
    if msg.token_length:
        header = (header << 8 * msg.token_length) | msg.token
    return header.to_bytes(CoAP.HEADER_LEN + msg.token_length, 'big')
```

3.3. CoAPMessage

Clasa specifica mesajului de tip CoAP, oferind un acces facil la toate campurile specifice acestui tip de mesaj (payload, token etc.). În același timp, oferă capabilitatea de a valida mesaje.

```python
class CoAPMessage:
    def __init__(self, payload: str, msg_type: int, msg_class: int, msg_code: int, msg_id: int,
                 header_version=0x1, token_length=0x0, token=0x0):
        self.payload = payload
        self.header_version = header_version
        self.msg_type = msg_type
        self.token_length = token_length
        self.msg_class = msg_class
        self.msg_code = msg_code
        self.msg_id = msg_id
        self.token = token

    def __str__(self) -> str:
        return f"""[VERSION]:\t{self.header_version}\n[TYPE]:\t\t{self.msg_type}\n[TKN LEN]:\t{self.token_length}
[CLASS]:\t{self.msg_class}\n[CODE]:\t\t{self.msg_code}\n[MSG ID]:\t{self.msg_id}
[TOKEN]:\t{hex(self.token) if self.token_length else ''}
[PAYLOAD]:\t{self.payload}\n"""
```

```python
    @classmethod
    def from_bytes(cls, data_bytes: bytes) -> 'CoAPMessage':
        header_bytes = data_bytes[0:4]
        header_version = (0xC0 & header_bytes[0]) >> 6
        msg_type = (0x30 & header_bytes[0]) >> 4
        token_length = (0x0F & header_bytes[0]) >> 0
        msg_class = (header_bytes[1] >> 5) & 0x07
        msg_code = (header_bytes[1] >> 0) & 0x1F
        msg_id = (header_bytes[2] << 8) | header_bytes[3]

        # Check if the header has reserved field values
        if header_version not in CoAP.VALID_VERSIONS:
            raise InvalidFormat("Message has incorrect CoAP header version")
        elif 9 <= token_length <= 15:
            raise InvalidFormat("Message has incorrect CoAP token length")
        elif msg_class in CoAP.CLASS_RESERVED:
            raise InvalidFormat("Message uses reserved CoAP message class")

        # Check special message types/classes/codes
        if (msg_class == CoAP.CLASS_METHOD and msg_code == CoAP.CODE_EMPTY) or msg_type == CoAP.TYPE_RESET:
            # This message must be EMPTY
            if not CoAPMessage.is_empty(msg_class, msg_code, token_length, data_bytes):
                raise InvalidFormat("Incorrect format for EMPTY CoAP message")
```

3.4. Server

Se ocupă în primul rand de realizarea legăturii cu clientul, folosindu-se de biblioteca *Socket*. Din momentul în care este pornit cu ajutorul butonului din interfața el “asculta” mesajele pe care le primește și le procesează, construind mesajul de raspuns pe care îl trimite înapoi clientului. De asemenea, sunt construite cateva mesaje pe care le afisam ulterior în interfața.

3.5. FSComponent

Tipul general al unui component din sistemul nostru de fișiere; din acesta extindem clasele File și Directory. Aceasta reține prin atribute date necesare precum părintele, calea curentă spre aceasta componenta sau nivelul acesteia în ierarhie.

3.6. File

Clasa corespunzătoare pentru un simplu fișier text din sistemul nostru de fișiere, pe langa atributele din FSComponent mai are și conținut.

3.7. Directory

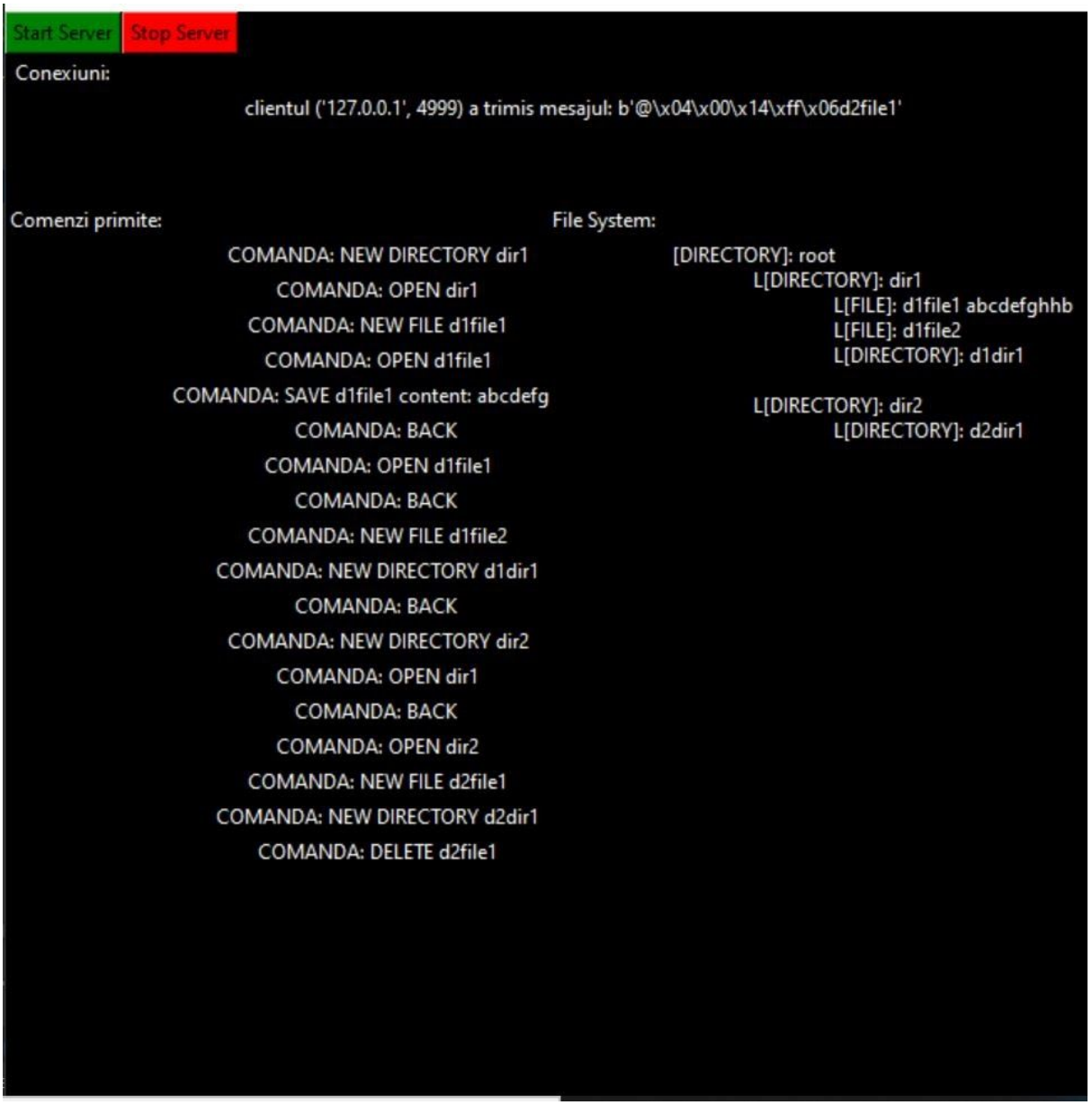
Clasa corespunzătoare pentru obiectele de tip director din sistemul de fișiere, avand un set de copii în care reținem urmașii lui, fie ei alte foldere sau simple fișiere.

3.8. DecodePayload

Clasa care răspunde de descifrarea comenzilor primite în incarcatura mesajelor de la client, dar și de execuția lor, respectiv realizarea acțiunilor respective în sistemul de fișiere.

4. Functionalitate

După lansarea în execuție a server-ului se creeaza directorul rădăcină root, se instantiaza un obiect de tipul DecodePayload și se așteaptă primirea mesajelor din partea clientului care sunt tratate pe rand, cat timp server-ul este pornit, prin metoda parsePayload. Aceasta returnează incarcatura mesajului răspuns, precum și clasa și codul acestuia. Apoi se începe compunerea restului mesajului răspuns. Dacă tipul mesajului primit este confirmabil, atunci tipul mesajului răspuns este de confirmare. În caz contrar, tipul noului mesaj va fi neconfirmabil. Campurile header_version, token_length și token sunt preluate de la mesajul primit, iar id-ul ramane acelasi în cazul mesajelor confirmabile. Mai departe, se creeaza obiectul de tip CoAPMessage si se transforma in bytes cu ajutorul metodei wrap, iar apoi se transmite înapoi către client cu funcția sendto specifica UDP. Comenzile realizate sunt bagate într-o coada, iar apoi sunt transmise către interfața unde sunt afișate. De asemenea, în interfața afisam și ierarhia de fișiere pe masura ce se actualizează.


Start Server
Stop Server
Conexiuni:
clientul ('127.0.0.1', 4999) a trimis mesajul: b'@\x04\x00\x14\xff\x06d2file1'
Comenzi primite:
COMANDA: NEW DIRECTORY dir1
COMANDA: OPEN dir1
COMANDA: NEW FILE d1file1
COMANDA: OPEN d1file1
COMANDA: SAVE d1file1 content: abcdefg
COMANDA: BACK
COMANDA: OPEN d1file1
COMANDA: BACK
COMANDA: NEW FILE d1file2
COMANDA: NEW DIRECTORY d1dir1
COMANDA: BACK
COMANDA: NEW DIRECTORY dir2
COMANDA: OPEN dir1
COMANDA: BACK
COMANDA: OPEN dir2
COMANDA: NEW FILE d2file1
COMANDA: NEW DIRECTORY d2dir1
COMANDA: DELETE d2file1
File System:
[DIRECTORY]: root
L[DIRECTORY]: dir1
L[FILE]: d1file1 abcdefghhb
L[FILE]: d1file2
L[DIRECTORY]: d1dir1
L[DIRECTORY]: dir2
L[DIRECTORY]: d2dir1