**Návrh a implementácia efektívneho komunikačného protokolu pre IoT meteorologické zariadenia**

Počítačové a komunikačné siete

**Filip Múdry**

AIS ID: 127504

Obsah:

1. Titulná strana ……………………………………….……. s. 0
2. Prehľad riešenia ………………………………………….. s. 2
3. Vývojové diagramy: Server …………..…………………. s. 3–5
4. Vývojové diagramy: Tester ……………..…………………s. 6–8
5. Správy v JSON + príklady ……………………………….. s. 8–13
6. Sekvenčné diagramy: UAT1 a UAT2 …………………..… s. 14–15
7. Zoznam použitých knižníc ………………………………...s. 16
8. Meranie efektivity prenášaných dát ……………………… s. 16–18
9. Záver …………………………………………………...… s. 19

# **2. Prehľad riešenia**

Systém pozostáva z dvoch programov komunikujúcich cez UDP:  
**Server** (centralizovaný príjemca a kontrolór aktivity) a **Tester** (klient simulujúci viaceré senzory). Výmena správ prebieha vo formáte JSON. Integrita dát je chránená kontrolným súčtom CRC32 vypočítaným z poľa data. Autorizácia odosielania dát prebieha pomocou **tokenu**, ktorý server pridelí pri registrácii senzora.

## Architektúra a roly

### Server

Naslúcha na UDP porte, prijíma správy a podľa typu vykonáva akciu.

Pri správe register overí identitu a typ senzora, vynúti politiku „jeden aktívny senzor na typ“, vygeneruje token a odpovie register\_ack.

Pri správe data kontroluje prítomnosť sensor\_id, token a správnosť CRC32 nad poľom data. Pri úspechu odpovie data\_ack, pri chybe vyžiada opätovné zaslanie (request\_resend).Periodicky monitoruje aktivitu registrovaných senzorov zasielaním activity\_check. Bez odozvy v limite vyhodnotí senzor ako DISCONNECTED; po prijatí activity\_ack zaznamená RECONNECTED.

### Tester

Pri štarte registroje všetky simulované senzory a ukladá si tokeny z register\_ack. V automatickom režime v intervale vytvára správy data s príznakom low\_battery podľa stavu, počíta CRC32 nad data a odosiela. Po data\_ack označí poslednú správu ako úspešne doručenú.V manuálnom režime umožňuje poslať presne definované merania. Pre UAT3 vie zámerne odoslať chybný rámec (nesprávna CRC pri rovnakom timestamp a data) a na request\_resend poslať korektnú verziu.

Na activity\_check odpovedá activity\_ack podľa pravidiel implementácie (napr. po treťom pingu pri pozastavenom senzore alebo keď je auto-odosielanie aktívne).

# A diagram of a company AI-generated content may be incorrect.Vývojové diagramy – Server

Obrázok 1 Server - prijímanie

Server nepretržite naslúcha na svojom UDP porte a čaká na príchod správ od testerov (senzorov). Po prijatí balíka si uloží adresu odosielateľa a obsah správy. Následne sa pokúsi správu prečítať ako JSON. Ak je správa neplatná alebo nečitateľná, zapíše do logu informáciu o chybe („neplatný JSON“) a pokračuje v čakaní na ďalší balík.

**Ak je správa platná, spracuje sa podľa poľa type:**

* **register** – Server overí identifikátor a typ senzora, zabezpečí, že pre daný typ existuje len jeden aktívny senzor, vygeneruje token a odošle odpoveď register\_ack.
* **data** – Server skontroluje prítomnosť sensor\_id a token, overí platnosť tokenu a správnosť CRC32 nad poľom data.
  + Pri chybe (neplatný token alebo nesprávna CRC) odošle request\_resend.
  + Pri správnych dátach ich spracuje, vypíše telemetriu (napr. čas, senzor, upozornenie na nízku batériu) a odošle data\_ack.
* **activity\_ack** – Označí daný senzor ako opäť aktívny a zaloguje „RECONNECTED“.
* A diagram of a company

  AI-generated content may be incorrect.**Iný typ správy** – Zaloguje ako neznámy a ignoruje.

Obrázok 2 Server - odoslanie

Server priebežne monitoruje všetky registrované senzory, aby overil ich dostupnosť a aktivitu. Každých približne 15 sekúnd prejde zoznam senzorov a tým, ktoré dlhšie neodoslali dáta, pošle správu activity\_check.

Ak odpoveď nepríde do stanoveného času, server zaznamená varovanie, zvýši počet pokusov a po desiatich neúspešných pingoch označí senzor ako **DISCONNECTED**.

Ak senzor odpovie správou activity\_ack, server jeho stav obnoví, vynuluje počítadlo pokusov a zaloguje **RECONNECTED**.

Okrem toho pri prijatí korektných dát (správa data) server odošle potvrdenie **data\_ack** a aktualizuje čas poslednej aktivity senzora.

# A diagram of a company AI-generated content may be incorrect.Vývojové diagramy - Tester

Obrázok 3 Tester odosielanie

Tester po spustení najprv inicializuje konfiguráciu odosielania a zaregistruje všetky senzory zaslaním správ typu **register**. Po úspešnej registrácii a prijatí tokenov začne pre každý aktívny senzor automaticky vytvárať dátové správy. Každá správa obsahuje identifikátor senzora, token, časovú pečiatku, namerané hodnoty, príznak nízkej batérie a kontrolný súčet **CRC32**.

V automatickom režime tester tieto správy pravidelne odosiela na server a čaká na potvrdenie **data\_ack**. Pri manuálnom režime je možné zadať hodnoty meraní ručne a po prijatí potvrdenia sa správa označí ako úspešne doručená.

Súčasťou logiky je aj test **UAT3**, pri ktorom tester zámerne odošle chybný rámec s nesprávnou CRC. Server na to reaguje správou **request\_resend**, po ktorej tester odošle pôvodnú správnu verziu dát. Tento cyklus sa opakuje, kým je proces odosielania aktívny.

A diagram of a company

AI-generated content may be incorrect.

Obrázok 4 Tester prijímanie

Tester v prijímacom cykle neustále počúva na sieti a spracúva správy, ktoré prichádzajú zo servera. Každý prijatý balík sa najskôr pokúsi prečítať ako JSON a podľa poľa **type** sa vykoná príslušná akcia.

Ak ide o správu **register\_ack**, tester uloží token, ktorý pridelil server konkrétnemu senzoru. Pri správe **data\_ack** označí posledne odoslané dáta ako úspešne doručené (presun z *pending* do *last\_sent\_ok*). Ak server pošle správu **request\_resend**, tester odošle znova korektnú verziu poslednej správy – buď vopred pripravený rámec z testu UAT3, alebo poslednú platnú správu z pamäte.

Na správu **activity\_check** tester reaguje podľa aktuálneho stavu. Ak je automatické odosielanie zapnuté, správa sa môže ignorovať. Ak je senzor pozastavený, tester odošle **activity\_ack** po treťom neúspešnom pokuse servera, čím potvrdí obnovenie spojenia. Všetky ostatné neznáme alebo neplatné správy sa iba zalogujú a cyklus pokračuje ďalej v prijímaní.

# Návrh jednotlivých správ v JSON formáte

## Register

**Účel:** Tester (senzor) sa registruje na server a žiada o pridelenie tokenu.  
**Odosiela:** Tester  
**Spracováva:** Server  
**Popis:** Slúži na identifikáciu senzora a jeho typu. Server overí, či už daný typ senzora nie je aktívny, vygeneruje token a pošle potvrdenie registrácie.

**Polia:**

* type – typ správy ("register")
* sensor\_id – identifikátor senzora (napr. "T001")
* sensor\_type – názov typu senzora
* timestamp – čas založenie / registrácie senzora

**Príklad:**

{ "type":"register", "sensor\_id":"T001", "sensor\_type":"ThermoNode", "timestamp":1730458200 }

## Register\_ack

**Účel:** Server potvrdzuje registráciu senzora a priraďuje mu token.  
**Odosiela:** Server  
**Spracováva:** Tester  
**Popis:** Každý senzor musí token uvádzať pri ďalších dátových správach. Token identifikuje konkrétne spojenie senzora so serverom.

**Polia:**

* type – "register\_ack"
* token – priradený identifikátor senzora
* timestamp – čas registracie senzora

**Príklad:**

{ "type":"register\_ack", "sensor\_id":"T001", "token":123456, "timestamp":1730458201 }

## Register\_denied

**Účel:** server odmieta registráciu senzora, pretože pre daný sensor\_type už existuje aktívny senzor (politika „jeden aktívny senzor na typ“).  
**Odosiela:** Server  
**Spracováva:** Tester  
**Kedy vzniká:** pri pokuse o registráciu nového senzora daného typu, kým je iný senzor rovnakého typu stále aktívny. Server vráti dôvod a identifikátor aktuálne aktívneho senzora.

**Polia:**

* type – vždy "register\_denied".
* reason – dôvod odmietnutia, napr. "type\_busy" keď je typ už obsadený.
* sensor\_type – typ, ktorý je obsadený (napr. "ThermoNode").
* active\_sensor\_id – identifikátor momentálne aktívneho senzora tohto typu.
* timestamp – čas vytvorenia správy (UNIX seconds).

**Príklad:**{

"type": "register\_denied",

"reason": "type\_busy",

"sensor\_type": "ThermoNode",

"active\_sensor\_id": "T001",

"timestamp": 1730458201

}

## Data

**Účel:** Odosielanie nameraných údajov zo senzora na server.  
**Odosiela:** Tester  
**Spracováva:** Server  
**Popis:** Obsahuje aktuálne merania, čas odoslania, stav batérie a kontrolný súčet CRC32, ktorý overuje integritu prenesených dát.

**Polia:**

* type – "data"
* sensor\_type – typ senzora
* sensor\_id – identifikátor senzora
* token – token pridelený serverom
* timestamp – čas odoslania (v sekundách)
* low\_battery – príznak nízkej batérie (true / false)
* data – objekt s meraniami (napr. teplota, vlhkosť…)
* crc32 – kontrolný súčet výpočítaný z poľa data

**Príklad:**

{

"type": "data",

"sensor\_type":"ThermoNode",

"sensor\_id": "T001",

"token": "A1B2C3",

"timestamp": 1730458200,

"low\_battery": false,

"data": { "temp\_c": 21.7, "humidity": 44.9 },

"crc32": "0xD3C1A4F9"

}

**Odpovede servera:**

Korektné dáta:

{ "type": "data\_ack", "sensor\_id": "T001" }

Poškodené dáta:

{ "type": "request\_resend", "sensor\_id": "T001" }

## data\_ack

**Účel:** Potvrdenie, že server úspešne prijal a spracoval dáta.  
**Odosiela:** Server  
**Spracováva:** Tester  
**Popis:** Tester si na základe tejto správy označí posledne odoslané dáta ako doručené.

**Polia:**

* type – "data\_ack"
* sensor\_id – identifikátor senzora
* timestamp – rovnaké ako predtým

**Príklad:**

{

"type": "data\_ack",

"sensor\_id": "T001"

"timestamp":1730458301

}

## request\_resend

**Účel:** Server žiada opätovné zaslanie dátovej správy, ak CRC alebo token nebol správny.  
**Odosiela:** Server  
**Spracováva:** Tester  
**Popis:** Tester po prijatí tejto správy odošle znovu poslednú korektnú správu s pôvodným timestampom a dátami.

**Polia:**

* type – "request\_resend"
* sensor\_id – identifikátor senzora

**Príklad:**

{ "type":"request\_resend", "sensor\_id":"T001", "timestamp":1730458301 }

## Activity\_check

**Účel:** Overenie, či je senzor stále aktívny (ping).  
**Odosiela:** Server  
**Spracováva:** Tester  
**Popis:** Server periodicky zasiela tieto správy všetkým registrovaným senzorom. Ak neodpovedajú, vyhodnotí ich ako odpojené.

**Polia:**

* type – vždy "activity\_check".
* sensor\_id – senzor, ktorého sa kontrola týka.
* timestamp – čas vytvorenia správy (UNIX seconds).
* attempt – poradové číslo kontrolného ping-u (napr. 1, 2, 3…).

**Príklad:**

{ "type":"activity\_check", "sensor\_type":"ThermoNode", "sensor\_id":"T001",

"token":123456, "timestamp":1730458350, "low\_battery":false, "attempt":3 }

## Activity\_ack

**Účel:** Odpoveď senzora na kontrolnú správu servera.  
**Odosiela:** Tester  
**Spracováva:** Server  
**Popis:** Tester ňou potvrdzuje, že je stále pripojený. Server po prijatí obnoví stav senzora a zapíše **RECONNECTED**.

**Polia:**

* type – "data"
* sensor\_type – typ senzora
* sensor\_id – identifikátor senzora
* token – token pridelený serverom
* timestamp – čas odoslania (v sekundách)
* low\_battery – príznak nízkej batérie (true / false)

**Príklad:**

{ "type":"activity\_ack", "sensor\_type":"ThermoNode", "sensor\_id":"T001",

"token":123456, "timestamp":1730458351, "low\_battery":false }

# A screenshot of a computer program AI-generated content may be incorrect.Sekvenčné diagramy pre komunikáciu UAT1 a UAT2

Obrázok 5 Sekvenčný diagram UAT1

A screenshot of a computer program

AI-generated content may be incorrect.

Obrázok 6 Sekvenčný diagram UAT2

# Zoznam použitých knižníc

**socket** – UDP komunikácia (server aj tester).

**json** – serializácia/deserializácia správ.

**threading** – paralelné odosielanie/monitoring popri CLI.

**random** – generovanie testovacích/náhodných hodnôt (tester) + generovanie tokenov

**time** – použitie na timestamp (ak niečo ešte využíva sleep alebo timestamp).

# Meranie efektivity prenášaných dát

### Cieľ

Cieľom merania je určiť, akú časť prenášaných bajtov tvoria skutočné užitočné dáta (merania) a akú časť zaberá réžia protokolu a nižších vrstiev (UDP, IP, Ethernet).  
Výsledkom je efektivita prenosu v percentách.

### Definície efektivity

**η\_data** – podiel užitočných dát (len obsah data)  
na celkovom počte bajtov prenesených po linke

 η\_data = |data| / |on\_wire|

**η\_msg** – podiel celej JSON správy  
na celkovom počte bajtov prenesených po linke

 η\_msg = |JSON| / |on\_wire|

|  |  |  |
| --- | --- | --- |
| **Vrstva** | **Položka** | **Veľkosť [B]** |
| Aplikačná | JSON správa | premenná |
| Transportná | UDP header | 8 |
| Sieťová | IPv4 header (bez options) | 20 |
| Linková | Ethernet II header | 14 |
| Linková | FCS | 4 |
| Linková | Preamble + SFD | 8 |
| Linková | IFG (inter-frame gap) | 12 |

**Celkové bajty „na linke“:**

|on\_wire| = 8 (Preamble + SFD)  
  + [14 + max(46, 20 + 8 + |JSON|) + 4] (Ethernet rámec)+ 12 (IFG)

Ak je (20 + 8 + |JSON|) menšie ako 46, doplní sa padovanie na 46 B.

### Postup merania

1. Zmeraj veľkosť JSON správy (bez medzier).
2. Skontroluj, či nie je potrebné padovanie (minimálna L2 payload = 46 B).
3. Spočítaj počet bajtov „na linke“ podľa vzorca vyššie.
4. Dosad’ do vzorcov η\_data a η\_msg.
5. Opakuj pre krátku a dlhšiu správu, aby bolo vidno rozdiel v efektivite.

### Modelové príklady

**A) Kratšia dátová správa**

|JSON| = 180 B |data| = 56 B  
20 (IP) + 8 (UDP) + 180 (JSON) = 208 ≥ 46 → bez padovania

on\_wire = 8 + (14 + 208 + 4) + 12 = **246 B**

η\_msg = 180 / 246 = **73.2 %**  
η\_data = 56 / 246 = **22.8 %**

**B) Dlhšia dátová správa**

|JSON| = 320 B |data| = 200 B

on\_wire = 8 + (14 + (20 + 8 + 320) + 4) + 12 = **386 B**

η\_msg = 320 / 386 = **82.9 %**  
η\_data = 200 / 386 = **51.8 %**

**Možnosti zlepšenia:**

* Zoskupovať viac meraní do jednej správy.
* Skracovať názvy polí v JSON alebo použiť binárny formát.
* Znížiť frekvenciu pingov a nepotrebných potvrdení.
* Odstraňovať z JSON medzery a zbytočné kľúče.

# Záver

Riešenie pozostáva z dvoch programov – **servera** a **testera**, ktoré medzi sebou komunikujú cez **UDP** pomocou správ vo formáte **JSON**. Server spracováva registrácie, prijíma dáta, overuje ich integritu pomocou **CRC32** a priebežne kontroluje aktivitu senzorov prostredníctvom správ activity\_check. Tester sa registruje, odosiela merania, reaguje na požiadavky servera a realizuje testovacie scenáre podľa zadania.

Všetky správy, sekvenčné aj vývojové diagramy zodpovedajú reálnemu priebehu komunikácie v kóde. Pri analýze efektivity sa potvrdilo, že najväčší podiel réžie tvoria hlavičky nižších vrstiev, zatiaľ čo efektivita prenosu rastie s dĺžkou dátovej časti. Riešenie spĺňa zadanie, je funkčné, prehľadné a pripravené na ďalšie rozšírenie.