Autor: Nikolas Knapik
Dátum: 25.11.2024

Dokumentácia pre P2P Protokol komunikácie cez UDP

Autor: Nikolas Knapik
Dátum: 25.11.2024

Obsah

Úvod: 2
Štruktúra hlavičkz protokolu: 2
Popis polí: 3
Implementácia hlavičky: 3
Fragmentácia: 4
Odosielanie dát: 4
Metóda na kontrolu integrity správy: 4
Výpočet CRC-16: 4
Porovnanie s CRC 32: 5
Metóda na zabezpečenie spoľahlivého prenostu dát: 5
Mechanizmus Stop-and-wait v mojom programe pracuje takto: 5
Výhody a nevýhody mechanizmu: 6
Metóda na udržanie spojenia (Keep-Alive) 6
Implementácia Keep-Alive: 6
Diagramy funkčnosti programu: 6
Záver: 7

Úvod:

Táto dokumentácia popisuje návrh protokolu P2P na prenos nát medzi dvoma uzlami pomocou UDP. Oba uzly fungujú zároveň ako klient a server a dokážu neustále posielať správy a súbory do 2 MB.

Cieľom je vytvoriť spoľahlivý prenos dát nad nespoľahlivým protkolom UDP.

Štruktúra hlavičkz protokolu:

Pre môj protokol som vytvoril vlastnú hlavičku, ktorá je pripojená kukaždému fragmentu. Obsahuje základné informácie potrebné na spracovanie správ a dát. Hlavička má pevnú veľkosť a skladá sa z nasledujúcich polí.

Autor: Nikolas Knapik
Dátum: 25.11.2024

Pole	Bajtový rozsah	Veľkosť (v bajtoch)
Typ správy	0	1
ID správy	1-2	2
Číslo fragmentu	3-4	2
Celkovo fragmentov	5-6	2
CRC	7-8	2

Popis polí:

- **Typ správy (1 bajt):** Určuje typ správy:
  - 01 : INIT (Inicializovanie spojenia)
  - 02 : SYN_ACK (Potvrdenie inicializácie)
  - 03 : ACK (Potvrdenie spojenia)
  - 04 : DATA (Dátový fragment)
  - 05 : ACK_DATA (Potvrdenie prijatia dátového fragmentu)
  - 06 : NACK_DATA (Nesprávne prijatie dátového fragmentu)
  - 07 : HEARTBEAT (S práva na kontrolu držania spojenia)
  - 08 : HEARTBEAT_ACK (Potvrdenie heartbeat správy)
  - 09 : FILE_INFO (Informácie o súbore)
- **Id správy (2 bajty) :** Identifikátor správy. Je unikátne generovaný pre každú správu.
- **Číslo fragmentu (2 bajty) :** Poradové číslo fragmentu v rámci celej správy (začína od 0). Umožňuje správne zloženie správy z prijatých fragmentov
- **Celkovo fragmentov (2 bajty) :** Udáva celkový počet fragmentov, ktoré tvoria celú správu. Príjamateľ vie, koľko fragmentov má očakávať.
- **CRC (2 bajty) :** Kontrolný súčet vypočítaný z dátového obsahu fragmentu pre overenie integrity.

Implementácia hlavičky:

Hlavička je v kóde vytvorená pomocou funkcie struct.pack a naformátovaná na reťazec ‘*! B H H H H*‘ , čo znamená :

- ! : Big-endian poradie bajtov (v sieťovom formáte)
- B : Unisgned char (1 bajt)
- H : Unsigned short (2 bajty)

Fragmentácia:

Fragmentácia je proces rozdelenia veľkej správy alebo súboru na mnešie časti, ktoré sa môžu efektívne prenášať cez sieť. Je dôležité správne nastaviť veľkosť fragmentu kedže je potrebná maximálna efektivita aj spoľahlivosť.
V mojom kóde som vypočítal maximálnu veľkosť fragmentu na 1463 kvôli veľkosti hlavičky.

Fragmenty sa v mojom kóde posielajú postupne. Vďaka stop-and-wait každý fragment čaká na potvrdenie doručenia kým program pokračuje ku ďalšiemu fragmentu.

Príjmateľ ukladá prijaté fragmenty podľa ich čísla. Po prijatí všetkých fragmentov ich spojí v správnom poradí, čím vytvorí pôvodnú správu alebo súbor.

Odosielanie dát:

V mojom programe sa na vstupe spýta užívateľa 1 pre správu a 2 pre súbor. Ak užívateľ zadá správu, program sa ho spýta, na správu. Následne sa vypíše do terminálu info o správe, ktorá sa posiela. Takisto po prijatí sa u príjmateľa vypíšu informácie o správe, ktorá bola obdržaná.

Ak si používateľ zvolí 2, opýta sa ho na cestu k súboru, ktorý chce poslať. Najskôr sa pošle správa FILE_INFO, kde sa nachádzajú informácie o súbore. Takisto po prenose sa vyprintujú informácie o danom súbore aj kde bol uložený.

Metóda na kontrolu integrity správy:

Na overenie integrity prenášaných dát som zvolil algoritmus CRC-16 (Cyclic Redundancy Check)

Výpočet CRC-16:

- **Iniciácia:**

Začneme s predvolenou hodnotou registra CRC, ktorá je zvyčajne 0xFFFF. Tento register bude obsahovať priebežný výpočet kontrolného súčtu.

- **Iterácia cez bajty dát:**

  Postupne sa spracováva každý bajt dát, ktoré chcem chrániť kontrolným súčtom.

  Každý bajt XOR-ujeme s horným bajtom registra CRC. Tým sa prvý bajt „premieša“ so začiatkom CRC

- **Bitové posuny:**

  Pre každý bajt dát sa opakuje cyklus 8 posunov (pre každý bit v bajte) kde sa posunieobsah registra CRC o jeden bit doľava (čím sa simuluje delenie). Ak sa najvyšší bit registra stane „1“, vykoná sa XOR s predvoleným polynómom (0x1021). Tento polynóm určuje vlastnost CRC algoritmu.

- **Maskovanie**:

  Po každom kroku sa uistíme, že register CRC zostáva v 16-bitovom rozsahu odstránením všetkých bitov nad 16-bitovou hranicou

- **Výsledok:**

  Po spracovaní všetkých bajtov dáj je hodnota v CRC registri výsledným kontrolným súčtom. Táto hodnota sa pridá k dátam pri ich odosielaní.

Porovnanie s CRC 32:

CRC 16 má oproti CRC 32 menej bajtov čo ale znamená rýchlejší výpočet. CRC 16 vyžaduje menej pamäťových zdojov, čo môže byť výhodné v našom prípade kedže máme jednoduchší systém.

Metóda na zabezpečenie spoľahlivého prenostu dát:

V protokole je implementovaný jednoduchý algoritmus na základe Stop-and-Wait aby som zabezpečil, že prenos dát medzi uzlami je spoľahlivejší.

Mechanizmus Stop-and-wait v mojom programe pracuje takto:

- **Odosielateľ:**

Rozdelí dáta na fragmenty podľa zvoleného fragmentačného limitu. Pošle jeden fragment a čaká na potvrdenie (ACK) od príjmateľa. Počas implementácie sa vyskytol problém kedy mi jeden fragment posielalo viac krát. Vyriešil som to použitím fronty ack_queue kde sa ukloží info o posledne odoslanom fragmente z ACK_DATA správy. Ak obdrží NACK_DATA ktoré sa kontrolujú pomocou crc, crc prepočíta a odošle fragment znova. Proces sa opakuje, kým program neobrdží správny ACK_DATA.

- **Príjmateľ:**
  Príjme fragment a overí jeho integritu pomocou CRC. Ak je fragment v poriadku, pošle ACK_DATA. Ak je fragment poškodený, pošle sa NACK_DATA .

Výhody a nevýhody mechanizmu:

Mechanizmus Stop-and-Wait je pomerne jednoduchý na implementáciu ale zároveň je spoľahlivý na prenos dát nad UDP protokolom.

Nevýhodou je nízka efektivita a rýchlosť pri vysokých latenciách alebo veľkých objemoch dát.

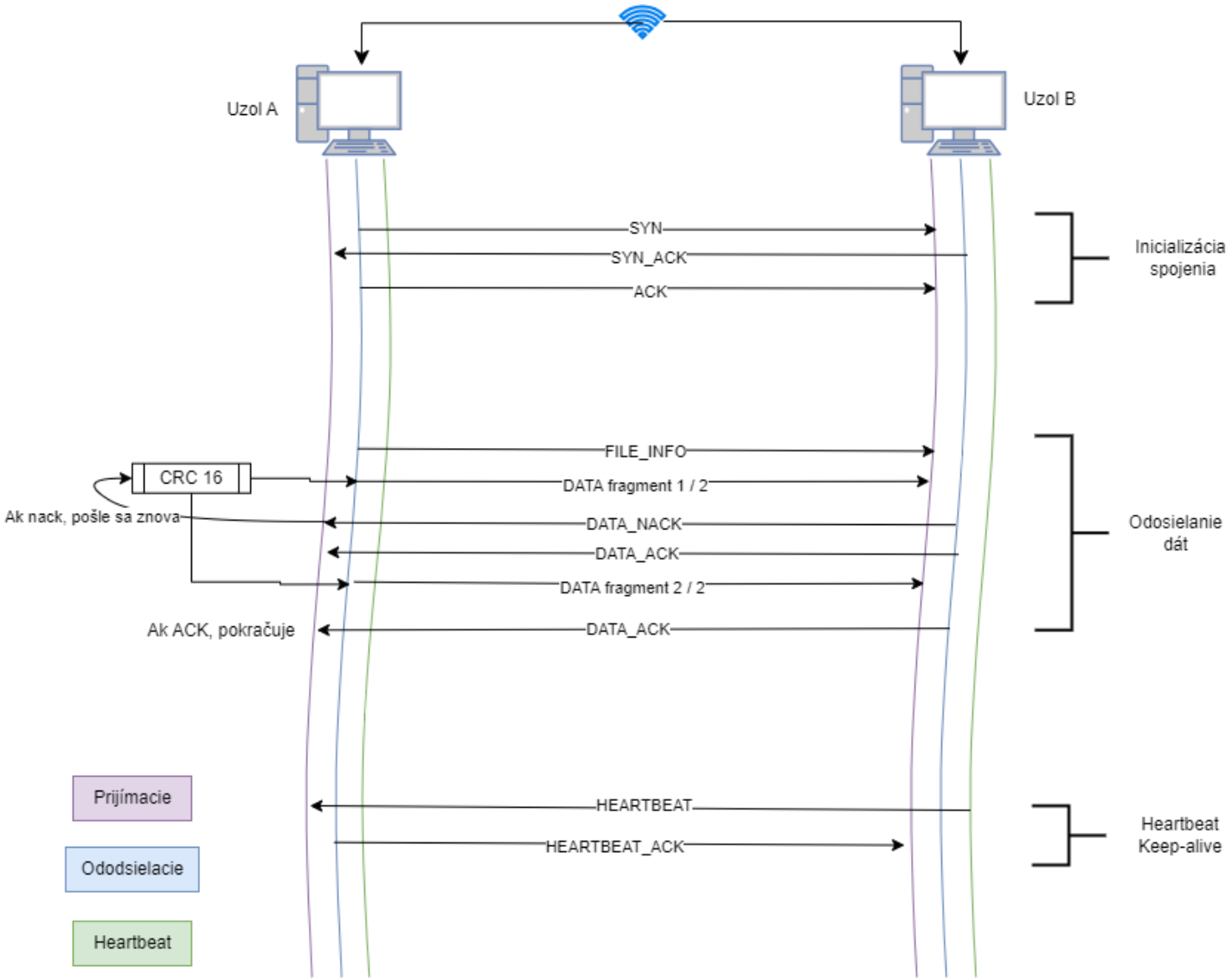
Metóda na udržanie spojenia (Keep-Alive)

Aby som zabezpečil, že spojenie medzi uzlami je stále aktívne, zaviedol som Keep-Alive mechanizmus.

Implementácia Keep-Alive:

- **Typy správ:** Ako typy správ používam 07 pre HEARTBEAT a 08 pre HEARTBEAT_ACK.
- **Proces:** Po inicializácií sa hanDshaku sa aktivuje vlákono heartbeat_thread, na ktorom sa každých 5 sekúnd pošle HEARTBEAT správa. Oba uzly posielajú správu a čakajú na odpoveď HEARTBEAT_ACK. Ak odpoveď neobdržia, inkrementuje sa počítadlo. Ak 3 krát neobdrží odpoveď, spojenie sa považuje za ukončené.

Diagramy funkčnosti programu:



Záver:

V tejto dokumentácií som predtavil návrh a implementáciu vlastného P2P protokolu, ktorý umožňuje spoľahlivý prenos správ a súborov medzi dvoma uzlami v lokálnej sieti.