PKS - Zadanie 2

Návrh riešenia

Maroš Hužička

ID: 120808

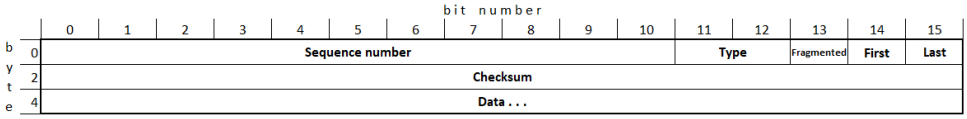
Utorok, 16:00

5. decembra 2023

Obsah

1 Protokol

Pre riešenie bol navrhnutý nasledujúci protokol:



- **Sequence number** - poradie správy v komunikácii. 0 je rezervovaná pre keep-alive správy. Ak sa dosiahne maximálna hodnota, následuje hodnota 1 a pokračuje číslovanie ďalej.
- **Type** - message/file/ACK/NACK
- **Fragmented** - indikuje, že správa predstavuje fragment väčšej správy. V prípade, že odosielaná správa je fragmentovaná, všetky fragmenty majú tento bit nastavený na 1.
- **First** - Prvý fragment; v prípade odosielania súboru data obsahujú názov súboru.
- **Last** - indikuje, že správa predstavuje posledný fragment.
- **Checksum** - checksum pre CRC.

- **Data** - dáta variabilnej dĺžky. Minimálna dĺžka dát je 0 (pošle sa iba hlavička) a maximálna, vzhľadom na to, aby sa odoslaný rámec ďalej nefragmentoval na linkovej vrstve, 1428 B (1500B - 60B (max IP hlavička) - 8B (UDP hlavička) - 4B (hlavička tohto protokolu))

1.1 Posielanie textových správ

V prípade, že uzol 1 posiela uzlu 2 textovú správu, uzol 1 najprv spracuje vstup z CLI - preformátuje do UTF-8, vypočíta checksum, prípadne rozdelí na viacero fragmentov... Následne pridá hlavičku s typom *message* a odošle správu na uzol 2.

Uzol 2 príjme správu a pomocou CRC zistí, či správa nie je poškodená. Ak nie, program vytlačí správu na konzolu a pošle ACK správu s rovnakým sequence number uzlu 1. Ak áno, pošle správu typu NACK (opäť s rovnakým sequence number) a čaká na opätovné prijatie správy.

Ak uzol 1 nezachytí ACK na svoju správu do určitého času (zatiaľ presne neurčený, no približne 2-10 sekúnd) alebo zachytí správu typu NACK, vyšle správu znova a čaká na ACK. Vykoná X pokusov o preposlanie a následne, v prípade neúspechu, prestane posielať správu a oboznámi používateľa, že odoslanie nebolo úspešné.

1.2 Posielanie súborov

Ak chce uzol 1 poslať súbor uzlu 2, pošle správu typu *file*. Dáta budú najprv obsahovať absolútnu cestu kam sa má súbor na cieľovom uzle uložiť následovanú znakom Line Feed (LF). Po tomto znaku nasleduje obsah súboru. Ďalšie prípadné fragmenty obsahujú iba obsah súboru.

1.3 Udržiavanie spojenia

Pre udržiavanie spojenia sa každých 5 sekúnd pošle správa so **Sequence number** 0 (Takzvaná keep-alive-message). Za odosielanie tejto správy je zodpovedný uzol, ktorý ako prvý poslal správu. Prijímateľ tejto správy odošle správu bez dát typu **ACK** taktiež so sequence number 0.

V prípade, že uzol nedostane odpoveď na keep-alive-message, odošle ešte niekoľko takýchto správ už v menšom časovom intervale a následne informuje používateľa o ukončení spojenia.

V prípade, že uzol, ktorý odosielal správu typu ACK nedostane keep-alive-message, rozhodne sa odoslať túto správu on. Ak získa ACK správu, ďalšie takéto správy neposiela.

Komunikáciu je možné ukončiť poslaním správy so sequence number 0 typu **NACK**, na čo druhý uzol odošle správu typu **ACK**.

1.4 potvrdenie prijatia správy

Na každú prijatú správu odoslanú Uzlom 1 odošle Uzol 2 správu typu **ACK** (acknowledge). Správa bude mať rovnaký sequence number ako pôvodná správa. V prípade, že Uzol 1 nedostane odpoveď na správu, bude sa niekoľkokrát pokúšať poslať správu znova.

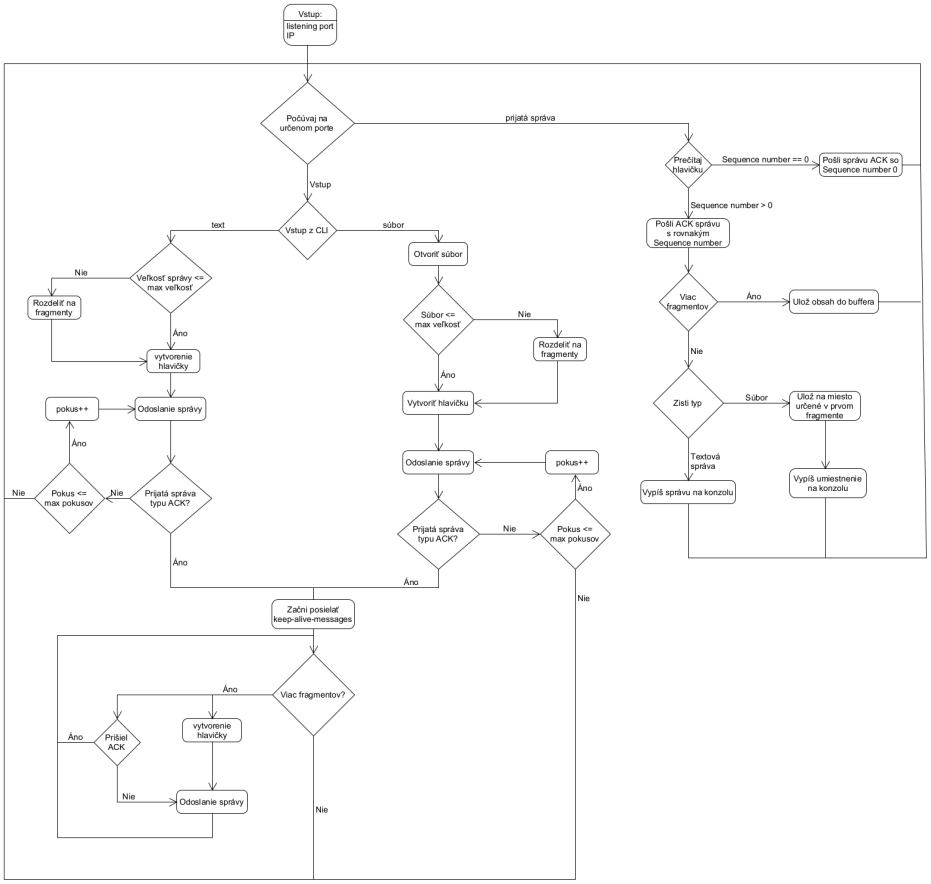
2 Kontrola prijatých dát a ARQ

Na kontrolu prijatých dát sa využíva **16-bitový Cyclic Redundancy Check (CRC)**. V programe určená 17-bitová konštanta sa použije na zakódovanie kontrolnej sumy. Najprv sa k Dátam pripíše 16 bitov s hodnotou 0. Ďalej sa pomocou bitovej operácie XOR vynuluje správa a kontrolná suma sa zapíše do pridaných bitov. Tie sa priradia do hlavičky do časti *Checksum*.

Následne prijímajúca strana použije rovnakú konštantu na zakódovanie správy, a ak sa posledných 16 bitov nevynuluje, potom pošle správu typu **NACK**, čo je signál pre vysielajúcu stranu, že správa sa nedoručila správne a je potrebné ju odoslať znova.

Nakoľko každá správa má svoj Sequence number, je možné vypýtať si odoslanie konkrétneho fragmentu, ktorý sa buď stratil počas komunikácie alebo sa doručil poškodený. Nie je potrebné posielať fragmenty, na ktoré bola prijatá ACK správa. Práve preto, ARQ metóda, ktorá sa implementuje bude založená na metóde **Selective Repeat**. Vysielač si bude ukladať v poli zoznam všetkých fragmentov (ich Sequence number) a ak na niektorý z nich príde správa **NACK** alebo po nejakom čase sa odpoveď nedostaví, znova pošle daný fragment.

3 Diagram spracovávania komunikácie



4 Návrh

Riešenie bude implemetované v programovacom jazyku **Python** s použitím modulu *socket*. Uzol v komunikácii je **vysielač a prijímač zároveň**, preto nie je nutné meniť túto funkcionalitu počas behu programu. Na každom z uzlov bežia 3 vlákna. Jedno vláknu slúži na počúvanie na porte určenom užívateľom. Ďalšie vlákno slúží na komunikáciu s programom - odosielanie správ druhému uzlu, riadiace správy pre program... Tretie vlákno ma za úlohu odosielať keep-alive-messages.

Pre nadviazanie spojenia je potrebné odoslať prvú správu (textovú alebo súbor). Následne sa spustí vlákno, ktoré odosiela keep-alive-správy. Tento postup však bude možné obísť špeciálnym príkazom, ktorý začne posielať tieto správy bez potreby poslať textovú správu alebo súbor.

Program je navrhnutý tak, aby súčastne bolo možné odosielať správy iba na jeden uzol a pre zmenu prijímateľa je potrebné ukončiť spojenie s aktuálnym prijímačom. Naopak prijímač je schopný prijímať dáta od viacerých vysielačov.

Uzol v komunikácii je reprezentovaný triedou:

```
class communication_node():
  def __init__(self, IP_address: str, listening_port: int) -> None:
    self.node_IP = IP_address
    self.listening_port = listening_port
    self.node_main()
```

Vytvorením objektu tejto triedy sa spustí funkcia **node_main**, ktorá riadi celú komunikáciu. Používateľ má možnosť zvoliť si, či chce iba počúvať, alebo aj odosielať vlastné správy (samozrejme, jedná sa len o posielanie vlastných textových správ a súborov, ACK a NACK správy je nevyhutné posielať).

5 Implementácia

5.1 Zmeny oproti návrhu

Riešenie prešlo mnohými zmenami počas doimplementácie, ktoré zásadne zmenili správanie a funkcionalitu programu.

5.1.1 Model klient-server

Namiesto riešenia Peer-to-Peer bol použitý model klient-server. Oba uzly však ostali implementované ako objekty rovnakej triedy, vďaka čomu je jednoduchšie prepínanie medzi funkciami vysielača a prijímača.

5.1.2 ARQ metóda

Taktiež metóda ARQ bola zmenená zo Selective Repeat na **Stop & Wait**. Vysielač pošle jednu správu typu MSG alebo FILE a očakáva správu typu ACK s rovnakým sequence number. Potom pošle ďalší fragment a pokračuje, kým neposlal všetky fragmenty.

Prijímač pri prijatí správy, ktorá je fragmentovaná najprv skontroluje, či sedí checksum. Ak nie, pošle správu typu NACK s rovnakým sequence, aký dostal a počka na opätovné prijatie správy. Ak je checksum správny, alebo nesedí sequence number (nie je viac ako predchádzajúce číslo), pošle ACK a počká na prijatie ďalšej správy, kým sa nebude sequence number zhodovať s očakávanu hodnotou (hodnota sequence nebude nikdy o viac ako 1 väčšia pri metóde Stop & Wait).

5.1.3 CRC

Použil sa polynóm **0x11021**, ktorý je odporúčaný spoločnosťou CCITT. Kontrolná suma sa počíta z hlavičky aj dát.

```
def __compute_checksum(self, Data: bytes) -> int:
  # ...

  divisor = self.polynomial
  divisor <<= data_bits - 17

  while number > 0xffff:
```

```
        number = number ^ divisor

        data_bits = number.bit_length()
        div_bits = divisor.bit_length()
        divisor >>= div_bits - data_bits

    return number
```

5.2 Trieda communication_node

Pri vytvorení objektu tejto triedy sa volá funkcia tejto triedy **node_main()**. Tá pracuje v nekonečnom cykle, ktorý dokáže ukončiť používateľ, čím sa prakticky vypne program. V tomto cykle si používateľ určí funkciu svojho uzla v komunikácií (vysielač/prijímač). Spustia sa relevantné thready (pre vysielača vlákno pre odosielanie správ a vlákno pre odosielanie keep-alive správ; pre prijímateľa vlákno pre prijímanie správ).

Následne hlavné vlákno prejde do funkcie **std_input()**. V tejto funkcii sa spracúva vstup z CLI. To je bližšie definovné v používateľskej príručke.

5.2.1 Vysielač

Používa funkciu *sender_node()*, ktorá si ako parametre pýta adresu a port prijímateľa. Následne počúva prichádzajúce ACK na healthchecky a ak dostane vstup a ten predstavuje definovaný typ správy, vypýta si od používateľa relevantné údaje (Správu, súbor na poslanie...). Ak je správa dlhšia ako veľkosť fragmentov, rozdelí sa správa na viacero fragmentov, z ktorých sa postupne vytvárajú protokoly.

Po poslaní správy program očakáva spätný acknowledge s rovnakým sequence number, aký odoslal. Ak po určitom čase ACK nepríde, prepošle fragment znova. Po 10 neúspešných pokusoch odoslania rovnakého fragmentu ukončí spojenie bez informovania druhej strany (nakoľko sa predpokladá, že úspešne nedorazí ani tá).

5.2.2 Prijímač

Používa funkciu *listener_node()*. Po prijatí správy na svoj port si uloží adresu a port odosielateľa pre prípadnú zmenu svojej funkcionality na vysielač. Nasledne spracuje správu podľa jej typu (Ak teda výpočet kontrolnej sumy - kontrolná suma datagramu = 0). Ak je checksum zlý, pošle odosielateľovi správu typu NACK (not acknowledge) s rovnakým sequence number:

```
# protocol.py

  class protocol:
    def __read_datagram(self, string: bytes) -> None:

      # Ak nesedi checksum, vrati objekt protocol s data_type None
      if len(string) > 4 and self.__compute_checksum(string [:2] +
          string[4:]) != int.from_bytes(string[2:4], "big"):
        self.sequence = int.from_bytes(string[:2], "big") >> 5
          self.data_type = None
```

```
            return

        self.__read_info(string)

# communication_node.py

    if recieve_protocol.data_type is None:
      print("Data recieved were corrupted (sequence",
          recieve_protocol.sequence,
           "). Asking for retransmission.\n")
      self.sock.sendto(protocol(recieve_protocol.sequence,
          "NACK").get_datagram(), client_address)
```

Ak je checksum správny, program spracuje všetky potrebné údaje do objektu triedy **protocol** a odošle správu typu ACK s rovnakým sequence number. Správa ACK sa pošle, aj keď sequence number prijatého datagramu nie je väčšia ako toho minulého (v prípade retransmisie, aby oznámil odosielateľovi, že tento fragment už je prijatý a môže sa poslať ďalší).

Ak prijímateľ dostane správu s bitom *First fragment* zapnutým, začne ukladať údaje do buffera, ktorý vypíše až keď príde správa s bitom *Last fragment*. Ak medzitým príde správa bez príznaku *Fragmented*, obsah buffera sa vymaže a vypíše sa nefragmentovaná správa.

Ak sa príjme súbor, po prijatí všetkých fragmentov sa obsah uloží do dočasného súboru **buffered_file_temp**. Spustí sa thread, ktorý očakáva na vstupe cestu, kde sa má daný súbor uložiť. Používateľovi sa zobrazí názov a prípona súboru, ktorý prijal. Pri zadavaní cesty je možné zadať absolútnu aj relatívnu cestu, avšak je potrebné zadať aj nový názov súboru spolu s príponou.

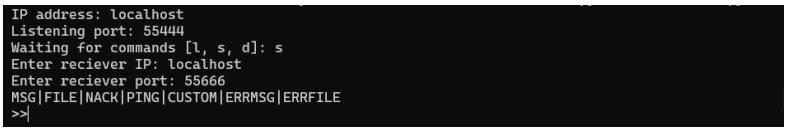
6 Používateľská príručka

Pri spustení programu je potrebné do konzoly vložiť IP adresu daného uzla a port, ktorý bude používať pre komunikáciu programu s druhým uzlom. Následne sa vytvorí objekt triedy *communication_node*. Po inicializácií objektu sa program opýta, akú úlohu v komunikácii bude zastávať. Možnosti sú nasledovné:

- **l** = listener,
- **s** = sender,
- **d** = koniec programu.

Obr. 1: Spustenie programu

Ak používateľ plánuje posielať správy alebo súbory, zvolí možnosť **s**. Následne je potrebné zadať adresu a port počúvajúceho uzla. Ak prebehlo všetko úspešne, používateľovi sa zobrazia možnosti poslania typu správy:



Obr. 2: Spustenie vysielača

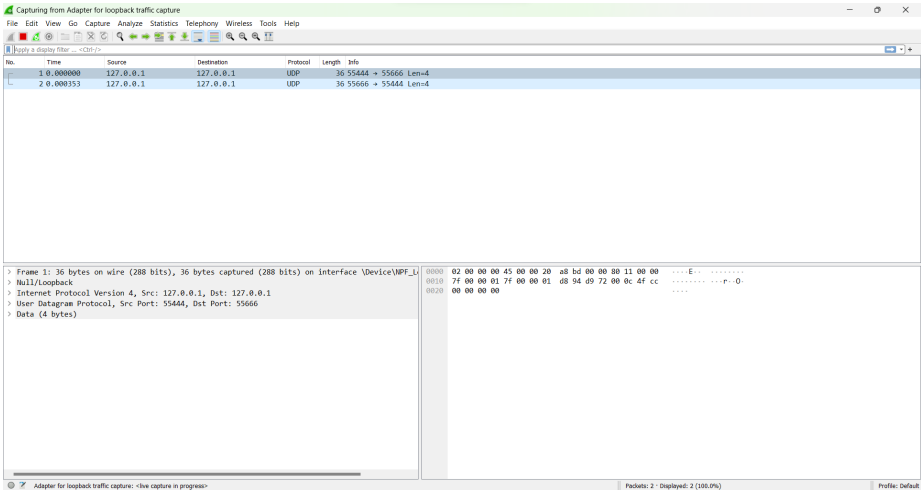
Samozrejme, pre komunikáciu sú potrebné oba uzly, preto je potrebné vytvoriť (či už na rovnakom systéme alebo použijeme iný počítač) prijímateľa. Vytvoríme novú inštanciu programu a do vstupu napíšeme **l**:

Obr. 3: Spustenie prijímača

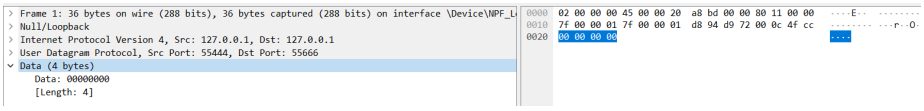
Stále však neprebieha žiadna komunikácia. Ak si otvoríme prostredie **Wireshark** a zadáme relevantné filtre, nevidíme žiadne rámce. Komunikácia nebude prebiehať, kým vysielač neodošle prvú správu, následkom čoho sa aktivuje vysielač keep-alive správ - **healthchecker**. Odošleme správu typu **PING**, ktorá umelo pošle keep-alive správu bez potreby healthcheckera. Prebehne cyklus poslania správy a aktivuje sa posielanie keep-alive správ healthcheckerom:

Obr. 4: Poslanie keep-alive správy ručne.

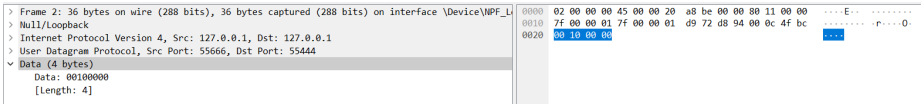
Ak klient zachytil správu typu ACK (acknowledge) na svoj datagram, do konzoly sa nám vypíše počet odoslaných fragmentov a celkový počet odoslaných bajtov (hlavička protokolu nad UDP a payload). Kedže sme odoslali iba keep-alive správu, počet odoslaných bajtov je 4, čo predstavuje hlavičku protokolu nad UDP. Keď sa pozrieme do Wiresharku, vidíme náš odoslaný datagram a acknowledge na našu správu s rovnakým sequence number:



Obr. 5: Odoslanie Pingu - pohľad z Wireshark-u. Môžeme vidieť odoslanie správy typu MSG so sequence number 0 (Táto kombinácia predstavuje keep-alive správu) a prijatie správy typu ACK so sequence 0 (odpoveď na keep-alive správu).



Obr. 6: Keep-alive správa, tzv. healthcheck.



Obr. 7: Acknowledge správa na healthcheck.

Po tejto operácií nám prijímajúci uzol už nehlási, že nedostáva žiadne keep-alive správy. Vo Wiresharku si možeme všimnúť, že **healthcheck sa posiela každých 5 sekúnd od poslania prvej správy**. My však chceme odoslať prijímateľovi textovú správu a otestovať, či správa prišla správne. Môžeme si nastaviť veľkosť jedného fragmentu pomocou príkazu **_FRAGMENTSIZE**. Správy začínajúce znakom _ a pokračujúce veľkými písmenami predstavujú príkazy pre program. Existujú tieto príkazy pre program:

- **_QUIT** - ukončí komunikáciu a vráti sa do výberu funkcie (vysielač/prijímač). Túto operáciu neoznámi druhej strane, pre správne ukončenie komunikácie je potrebné odoslať správu typu NACK so sequence number 0.
- **_SAVE** - pri zadaní lokácie uloží súbor na zadané miesto na disku.
- **_FRAGMENTSIZE** - určí veľkosť jedného fragmentu pri odosielaní dát

- **_CHANGE** - odošle žiadosť o zmenu funkcii v komunikácii druhej strane **(switch)**
- **_WHOAMI** - vypíše aktuálnu funkciu v komunikácii (sender/listener)

Pri spustení programu sa maximálna veľkosť fragmentu nastaví na maximálnu hodnotu (1428). Pre overenie zadáme do vstupu *_FRAGMENTSIZE* bez akejkoľvek hodnoty. Následne zadáme rovnaký príkaz s nami požadovanou hodnotou, ak je hodnota mimo rozsah, zmena sa nepodarí. Nastavíme veľkosť fragmentu na 1 bajt a odošleme textovú správu *Ahoj, server!*:

```
MSG|FILE|NACK|PING|CUSTOM|ERRMSG|ERRFILE
>>_FRAGMENTSIZE
Current maximum fragment size is 1428
_FRAGMENTSIZE 0
Invalid Value. Must be between 1 and 1428. Syntax is: _FRAGMENTSIZE 123
_FRAGMENTSIZE 1
Maximum fragment size is now 1
MSG

>> Enter message to send
>> Ahoj, server!
Total fragments sent: 13
Size of fragments: 5 B (Size of the last fragment was 5 B)

MSG|FILE|NACK|PING|CUSTOM|ERRMSG|ERRFILE
>>
```

Obr. 8: Zmena veľkosti fragmentu a odoslanie fragmentovanej správy.

Ako vidíme, správa sa odoslala v 13 fragmentoch (dĺžka nášho textu bola 13 znakov). Výstup je možné vidieť aj na strane prijímača:

```
Recieving from: 127.0.0.1 on port: 55444
Fragments recieved: 13  | Total bytes: 13 (payload only)
Size of fragments: 5 B (Size of the last fragment was 5 B)
Data:
 Ahoj, server!

```

Obr. 9: Prijatie textovej správy.

Správa bola prijatá. Teraz chceme odoslať súbor zo strany vysielača klientovi. Na to je potrebné vymeniť funkcie uzlov v komunikácii pomocou príkazu *_CHANGE*. Následne bude možné odosielať správy z opačnej strany:

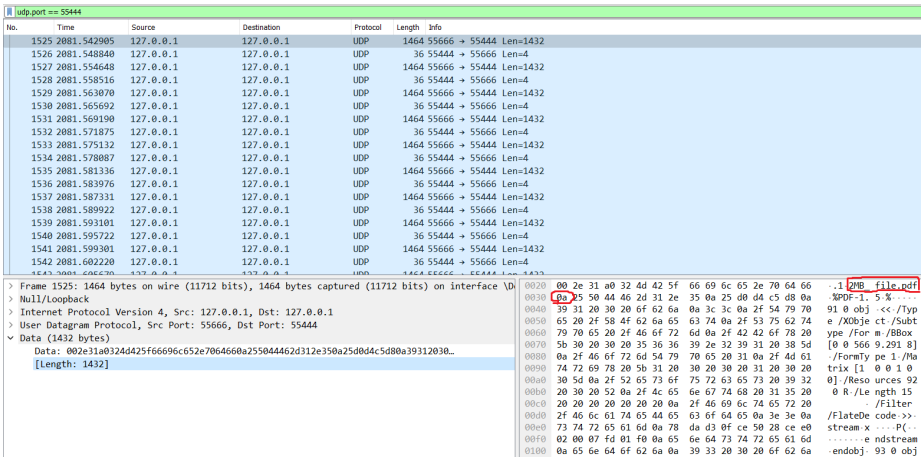
```
MSG|FILE|NACK|PING|CUSTOM|ERRMSG|ERRFILE
>>The listener asked to become sender (Press ENTER to continue).

Recieved file: 1519 fragments; 2168635 bytes recieved (payload only)
Fragment size: 1432 B (Last fragment was 935 B)
Where to save it? Its file name was 2MB_file.pdf
>> _SAVE C:\Temp\received.pdf
File saved, it's location is "C:\Temp\received.pdf"
```

```
_CHANGE
Changing to sender.
MSG|FILE|NACK|PING|CUSTOM|ERRMSG|ERRFILE
>>FILE

>> Enter full path to the file
>> C:\Temp\2MB_file.pdf
Sending 2168622 B
Total fragments sent: 1519
Size of fragments: 1432 B (Size of the last fragment was 935 B)

MSG|FILE|NACK|PING|CUSTOM|ERRMSG|ERRFILE
>>
```

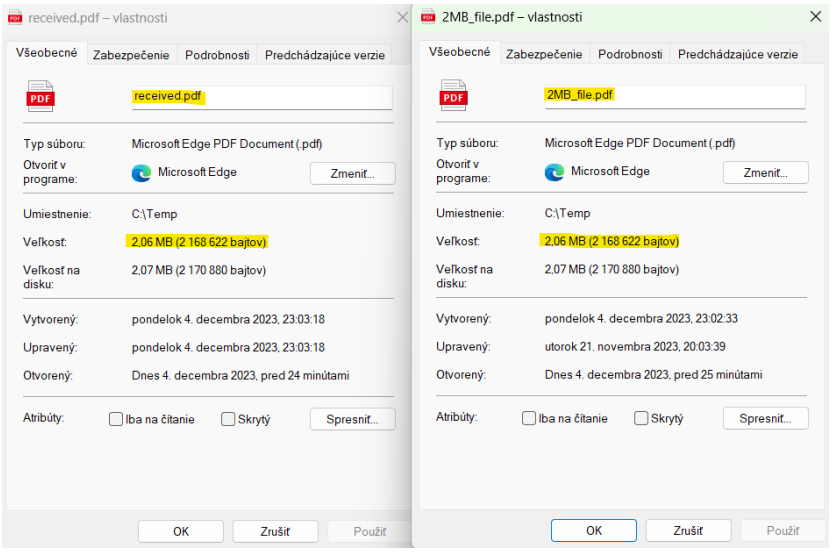
Obr. 10: Switch a odoslanie súboru. Na ľavej strane obrázku vidíme, ako sa zmenil vysielač na prijímač a uložil súbor na miesto určené používateľom. Na pravej strane vidíme nový vysielač, ako poslal súbor pôvodnému klientovi.

Môžeme si všimnúť, že všetky metadáta o veľkostiach a počtu fragmentov sa zhodujú, až na veľkosť celkového počtu odoslaných/prijatých dát. To nie je

chyba, vysielač zobrazuje **veľkosť súboru**, ktorý posiela, prijímač však ukazuje **celkový počet prijatých bajtov** (počítajú sa iba správne prijaté fragmenty). Prijímač prijíma najskôr názov programu s príponou, až potom príjme dáta súboru.

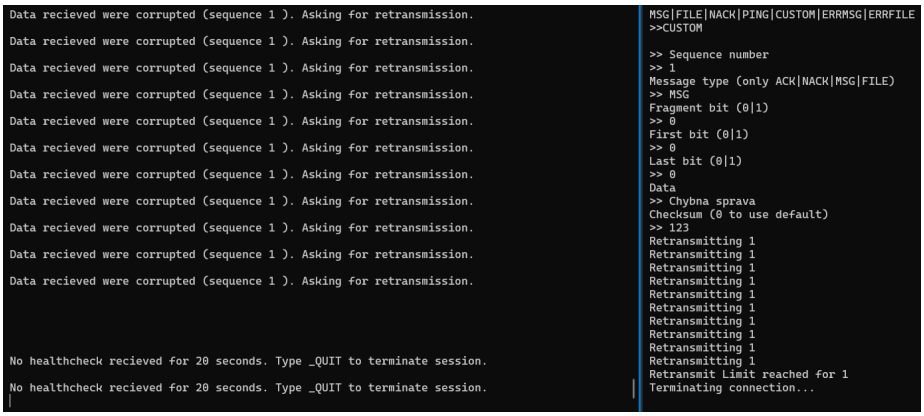


Obr. 11: V prvom fragmente sa nachádza názov súboru nasledovaný znakom nového riadku.



Obr. 12: Ak porovnáme veľkosť odoslaného a prijatého súboru, ich veľkosti sú rovnaké.

Program umožňuje poslať aj správu obsahujúcu chybu. Zvolíme možnosť CUSTOM pri výbere typu správy a zadáme vlastné údaje spolu s chybným checksum.



Obr. 13: Správa s chybným checksumom sa poslala niekoľko krát a nakoniec to vysielač vzdal.

Vysielač ukončil komunikáciu, avšak nenotifikoval druhú stranu (nakoľko predpokladal, že by správne neprišla ani správa pre ukončenie). Ak chceme ukončiť komunikáciu notifikovaním druhej strany, odošleme správu typu NACK. Odošle sa správa typu NACK so sequence number 0. Vysielací uzol príjme ACK správu a ukonči posielanie. Zadáme príkaz *_QUIT* a znova sa pripojíme ako vysielač, pošleme nejakú správu nasledujúcu správou NACK:

Obr. 14: Správne ukončenie komunikácie.

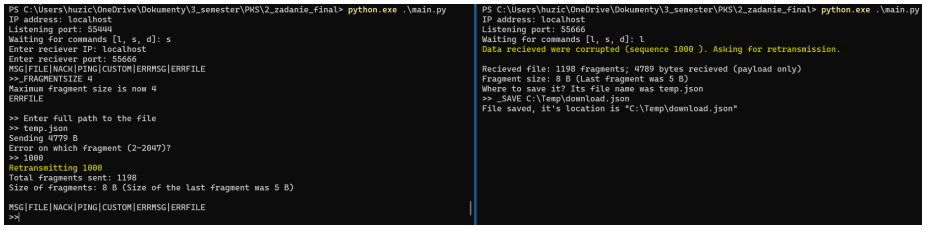
7 Testovanie - poslanie súboru s chybným fragmentom

V tomto scenári sa bude testovať odoslanie súboru s názvom *temp.json*, ktorého veľkosť je 4 779 bajtov. Zároveň však 1000. fragment pošleme s chybnými dátami a checksumom. Prijímač by to mal zachytiť. Vysielač pošle 1000. fragment s dátami *AAA*, vďaka čomu ho bude možné ľahko rozlíšiť v prostredí Wireshark od ostatných fragmentov.

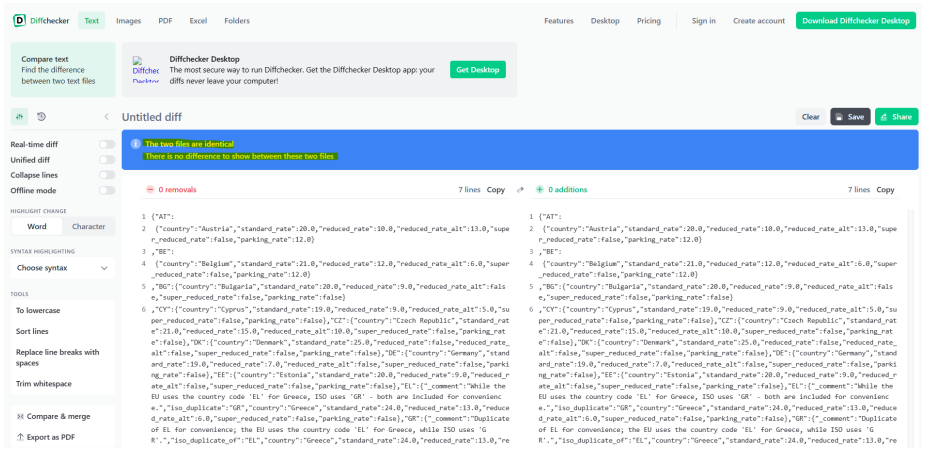
Pre odoslanie súboru s chybným fragmentom použijeme možnosť **ERRFILE**, určíme cestu k súboru (stačí relatívna, nakoľko sa nachádza v rovnakom adresári ako náš program) a zadáme sequence number chybného fragmentu.

V programe môžeme vidieť, že prijatý nesprávny fragment bol znova preposlaný správne. V prostredí Wireshark môžeme vidieť fragment so sequence number 1000 preposlaný dvakrát.

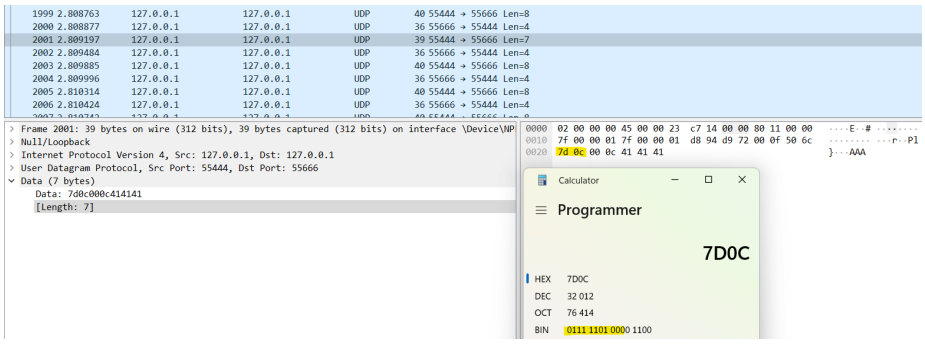
Dokázali sme, že program si vie poradiť s poškodeným fragmentom.



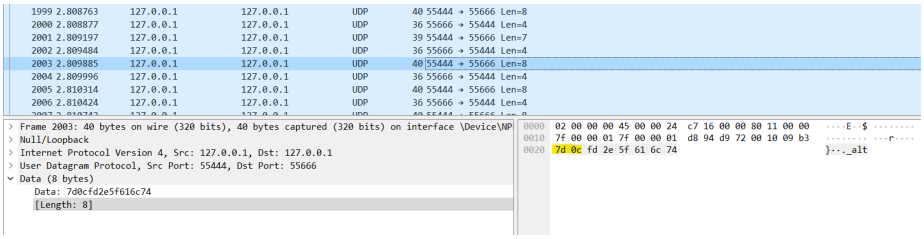
Obr. 15: Testovací scenár - výstup programu.



Obr. 16: Testovací scenár - neexistuje rozdiel medzi poslaným a prijatým súborom (porovnané v https://www.diffchecker.com/text-compare/).



Obr. 17: Zlý fragment. Prvých 11 bitov predstavuje sequence number, a to sa rovná 1000 v decimálnom tvare. Data sú AAA.



Obr. 18: Preposlaný fragment so správnymi dátami a checksumom, s rovnakým sequence number.