Slovenská technická univerzita Fakulta informatiky a informačných technológií Ilkovičova 3, 842 16 Bratislava 4

UDP komunikátor

2020/21, ZS

Matej Delinčák



Návrh protokolu

Vlastná hlavička

Môj protokol bude mať dve rozličné hlavičky. Informatívny paket a dátový paket. Informatívny paket bude vyzerať nasledovne:

Bit Number 111111111122222222233 01234567890123456789012345678901

INFORMATIVE PACKET

Туре	Packet number / Total packets (optional)
	Name of file (optional)

Pole *Type* bude obsahovať flagy o akú správu ide a bude mať veľkost 1B:

Strana klient bude používať:

"0" – začatie komunikácie a nepoužijem ostatné polia

"1" – keep alive správa a nepoužijem ostatné polia

"2" – bude poslaná správa a použijem pole Total packets

"3" – bude poslaný súbor a použijem obe polia *Total packets* ako aj *Name of file*

Strana server bude používať:

"4" – paket prišiel v poriadku a použijem len pole Packet Number

"5" – paket neprišiel v poriadku a použijem len pole Packet Number

"6" – všetky pakety sú v poriadku a nepoužijem ostatné polia

V poli *PacketNumber/Total packets* bude počet paketov, alebo poradové číslo paketov, ktoré budú poslané. Použil som 3B, práve preto, lebo máme vedieť poslať 2MB súbor. To je približne 2 milióny bajtov (ak by veľkosť fragmentu bolo 1B). A v *Name of file* poli bude názov súboru.

Na obrázku som označil dve polia ako optional. To preto, lebo keď posielam začatie komunikácie a keep alive, tak ich nepotrebujem použiť. Ak posielam správu, použijem pole **Total packets,** ale nie *Name of file*. A ak posielam súbor, tak použijem všetky tri.

Ďalej opíšem dátový paket:

Bit Number
1111111111222222222233
0123456789012345678901

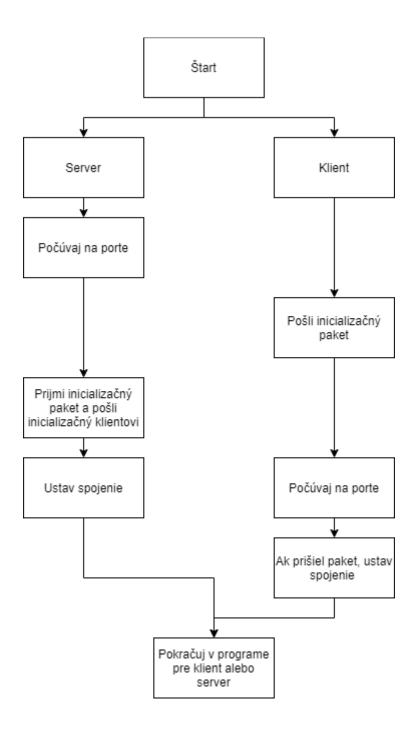
DATA PACKET

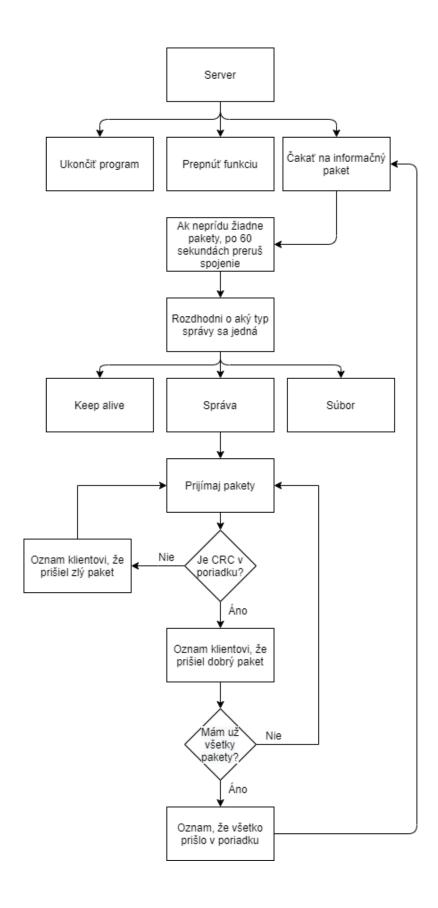
Packet number	CRC .						
CRC	Data						

V poli *Packet number* bude poradie paketu. 3B preto, lebo ako som už spomenul vyššie, môžem mať 2 milióny paketov. V poli *CRC* – veľkosť 4B – sa bude nachádzať zvyšok po CRC metóde. To ako funguje, opíšem neskôr vo vlastnej sekcií. V poli *Data* budú už samotné dáta. Maximálna veľkosť je 1465B. Pretože nechceme aby sa fragmentovalo na linkovej vrstve, treba veľkosť nastaviť na 1526B – 26B – 20B – 8B a – 7B kvôli mojej hlavičke.

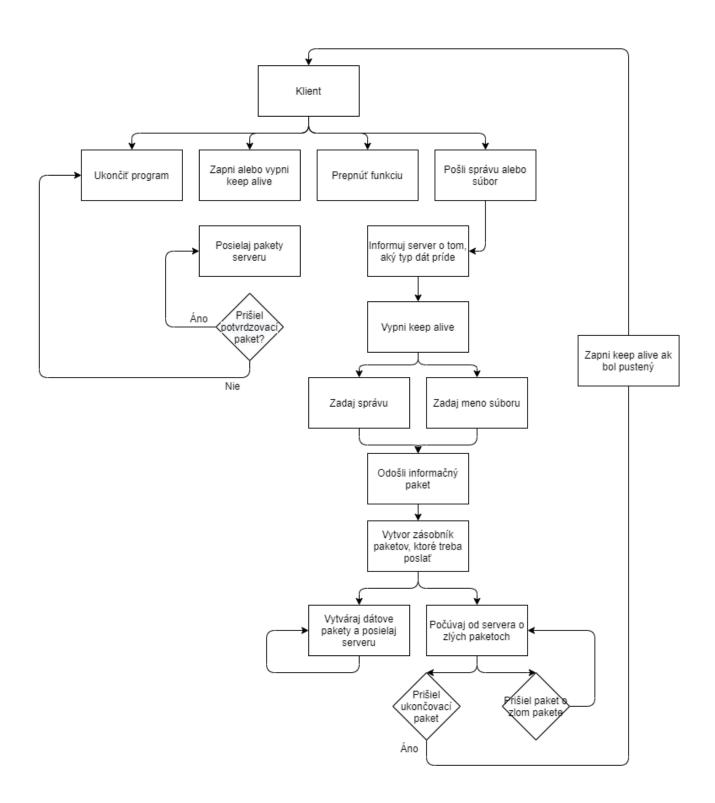
Message sequence diagram

V nasledujúcich diagramoch ukážem ako bude fungovať môj štart programu a samotné funkcie pre klient a server. Tieto diagramy nemusia byť vo finálnom odovzdaní rovnaké, môžu sa jemne líšiť.





Klient pri výbere si bude môcť vybrať aj rozličnú veľkosť fragmentu.



ARQ metóda

Pre môj návrh som si vybral Selective ARQ metódu. Celá táto metóda spočíva v tom, že klient bude posielať zaradom dátové pakety zo zásobníka. Druhý thread zároveň počúva na odozvy od serveru. Server počúva a prijíma pakety. Ak prijme paket, ktorý je v poriadku, odošle klientovi správu o dobrom prijatí. Ak je paket zlý, pošle klientovi správu s číslom zlého paketu.

Klient teda tieto správy prijíma a rozhoduje sa na ich základe. Ak prišla pozitívne ACK, tak si zapíše, že tento paket prišiel v poriadku. Ak príde negatívne ACK, tak tento paket znova zaradí do zásobníka na odoslanie. Ak už všetko poslal a nemá všetky správy potvrdené, pošle pakety, ktoré asi ešte chýbajú serveru.

Na koniec, keď server má všetky pakety, pošle globálne ACK klientovi a ten prestane posielať ďalšie dáta.

Checksum algoritmus

Ako algoritmus pre kontrolu, či dáta neprišli porušené, použijem metódu CRC. A to konkrétne funkciu *crc32()* z knižnice *zlib*. Túto metódu som skúšal naprogramovať sám, len bola veľmi veľmi pomalá, tak som sa rozhodol pre funkciu z knižnice. A teda aby som ju mohol použiť, opíšem ako funguje.

vstup	0	1	0	0	0	0	0	1																																	
obrateny vstup	1	0	0	0	0	0	1	0																																	
vstup + 32 nul	1	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	
0xFFFFFFF	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	0	0	0	0	0	0	0	0	
XOR	0	1	1	1	1	1	0	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	0	0	0	0	0	0	0	0	
polynom		1	0	0	0	0	0	1	0	0	1	1	0	0	0	0	0	1	0	0	0	1	1	1	0	1	1	0	1	1	0	1	1	1							
	0	0	1	1	1	1	0	0	1	1	0	0	1	1	1	1	1	0	1	1	1	0	0	0	1	0	0	1	0	0	1	0	1	1	0	0	0	0	0	0	
			1	0	0	0	0	0	1	0	0	1	1	0	0	0	0	0	1	0	0	0	1	1	1	0	1	1	0	1	1	0	1	1	1						
	0	0	0	1	1	1	0	0	0	1	0	1	0	1	1	1	1	0	0	1	1	0	1	1	0	0	1	0	0	1	0	0	0	0	1	0	0	0	0	0	
				1	0	0	0	0	0	1	0	0	1	1	0	0	0	0	0	1	0	0	0	1	1	1	0	1	1	0	1	1	0	1	1	1					
	0	0	0	0	1	1	0	0	0	0	0	1	1	0	1	1	1	0	0	0	1	0	1	0	1	1	1	1	1	1	1	1	0	1	0	1	0	0	0	0	
					1	0	0	0	0	0	1	0	0	1	1	0	0	0	0	0	1	0	0	0	1	1	1	0	1	1	0	1	1	0	1	1	1				
	0	0	0	0	0	1	0	0	0	0	1	1	1	1	0	1	1	0	0	0	0	0	1	0	0	0	0	1	0	0	1	0	1	1	1	0	1	0	0	0	
						1	0	0	0	0	0	1	0	0	1	1	0	0	0	0	0	1	0	0	0	1	1	1	0	1	1	0	1	1	0	1	1	1			
	0	0	0	0	0	0	0	0	0	0	1	0	1	1	1	0	1	0	0	0	0	1	1	0	0	1	1	0	0	1	0	0	0	0	1	1	0	1	0	0	
0xFFFFFFF									1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	
XOR	0	0	0	0	0	0	0	0	1	1	0	1	0	0	0	1	0	1	1	1	1	0	0	1	1	0	0	1	1	0	1	1	1	1	0	0	1	0	1	1	
obrateny vysledok									1	1	0	1	0	0	1	1	1	1	0	1	1	0	0	1	1	0	0	1	1	1	1	0	1	0	0	0	1	1	0	1	=CR(

Na vstupe mam pre príklad vstup "A", ten prevediem na číslo v binárnej sústave a zrkadlovo ho obrátim. Pridám k dátam na koniec 32 núl a na takýto vstup použijem XOR s hodnotou 0xFFFFFFFF. Teraz prichádza hlavný cyklus crc, kde XORujem polynóm s medzivýsledkom. Keď prídem na koniec výsledok zase zoXORujem s 0xFFFFFFFF a zrkadlovo obrátim. Výsledok je hodnota CRC.

Funkcia zlib.crc32() používa polynóm $x^{32} + x^{26} + x^{23} + x^{22} + x^{16} + x^{12} + x^{11} + x^{10} + x^8 + x^7 + x^5 + x^4 + x^2 + x^1 + 1$. Čo je v binárnej sústave 1 0000 0100 1100 0001 0001 1101 1011 0111.

Keep alive metóda

Ak klient pustí keep alive, tak každých 30 sekúnd sa pošle správa serveru. Ten keď obrdží túto správu, pošle ACK klientovi. Týmto sa bude udržiavať spojenie. Ak ale nepríde ACK, tak to klient vyhodnotí ako uzavretie spojenia. Na druhej strane server uzaviera spojenie, ak do 60 sekúnd nepríde nič.