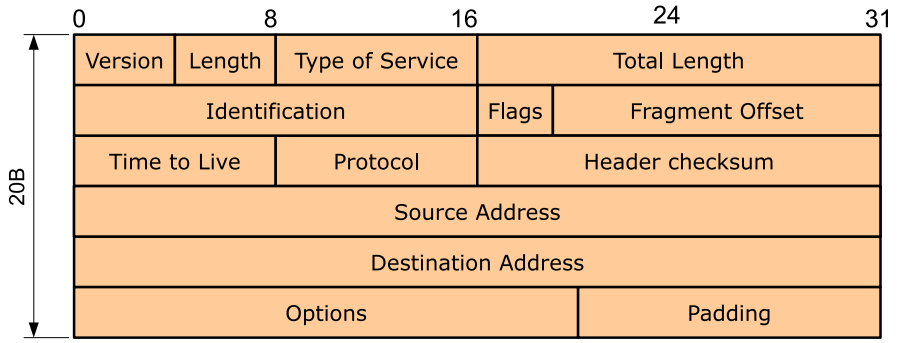
4 Komunikácia na báze TCP/IP protokolu

4.1 IP paket a jeho hlavička

TCP/IP paket je možné znázorniť ako

Obr. 3: Zloženie TCP/IP paketu (segmentu)

kde štruktúra hlavičky IP paketu je definovaná ako



Obr. 4: Štruktúra TCP/IP segmentu - hlavička IP paketu (segmentu)

VERS - (Version, 4 bity) definuje verziu IP protokolu, súčasná verzia ma poradové číslo 4

HLEN - (Header Length, 4 bity) udáva dĺžku záhlavia v 32 bytových slovách, nevyužité slabiky sú označované ako výplň

SERVICE - (Type of Service, 4 bity) udáva požiadavku na prioritu IP paketu (0-7), požadované minimálne oneskorenie, požadovanú vysokú kapacitu a požadovanú vysokú spoľahlivosť. Tieto požiadavky dovolia vybrať najvýhodnejšiu cestu, Internet však ich splnenie negarantuje.

TLENGTH - (Total length, 16 bitov) udáva celkovú dĺžku IP paketu (fragmentu) v slabikách, zahrňuje záhlavie aj dáta, najväčšia možná dĺžka IP paketu je 65535 slabík

IDENT - (Identification, 16 bitov) identifikuje IP paket a podporuje fragmentáciu

FLAGS - tri 1-bitové príznaky, podporujú fragmentáciu (zákaz fragmentacie, identifikácie posledného fragmentu v IP pakete)

OFFSET - (Fragment offset, 13 bitov) relatívna adresa fragmentu v Ip pakete v násobkoch ôsmich slabík

TTL - (Time to Live, 8 bitov) udáva dobu života IP paketu v sekundách, každá brána znižuje hodnotu o jedničku, pri nule je IP paket (fragment) likvidovaný.

PROTOCOL - (16 bitov) identifikácia protokolu vyššej vrstvy (TCP, UDP)

CHECKSUM - (Header Checksum, 16 bitov) kontrolný súčet záhlavia paketu (fragmentu)

SOURCEAD - (Source Address, 32 bitov) IP adresa odosielateľa IP paketu

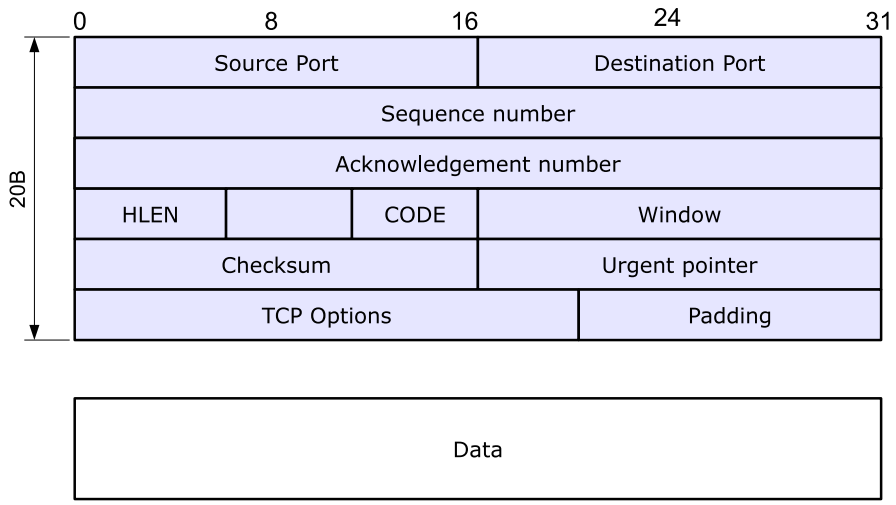
DESTAD - (Destination Address, 32 bitov) IP adresa adresáta IP paketu

OPTIONS - pripadné riadiace a ladiace funkcie, meranie v sieti.

PADD - (Padding) výplňové nulové slabiky.

4.2 TCP paket a jeho hlavička

Štruktúra TCP paketu a dát je nasledovná



Obr. 5: Štruktúra TCP/IP segmentu - hlavička TCP paketu a dáta

SRCPORT - (Source Port, 16 bitov) identifikácia socketu odosielateľa

DESTPORT - (Destination Port, 16 bity) identifikácia socketu adresáta

SWQNO - (Sequence Number, 32 bitov) poradové číslo (prvého znaku segmentu)

ACKNO - (Acknowledgement Number, 32 bitov) potvrdzovacie číslo (očakávaný znak)

HLEN - (Header length alebo Data Offset) dĺžka hlavičky v tridsaťdvabitových slovách

CODE - (6 bitov) riadiace bity URG, ACK, PSH, RST, SYN, FIN

WINDOW - (6 bitov) špecifikuje šírku prijímacieho okna vysielača

CHECKSUM - (16 bitov) kontrolný súčet TCP segmentu (vrátane pseudohlavičky IP)

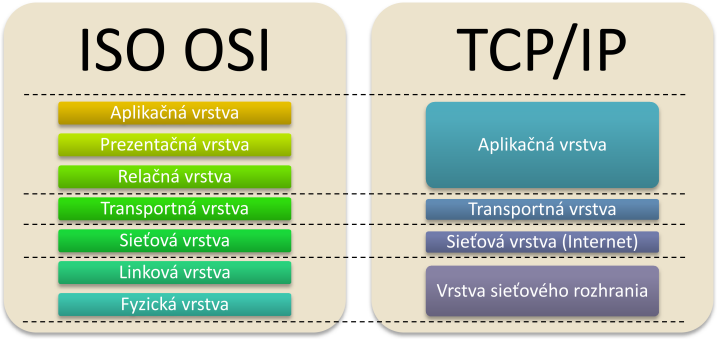
URGPNT - (16 bitov) kontrolný súčet TCP segmentu (- označenie konca urgentných dát)

OPTIONS - pripadné riadiace a ladiace funkcie, meranie v sieti.

PADD - (Padding) výplňové nulové slabiky.

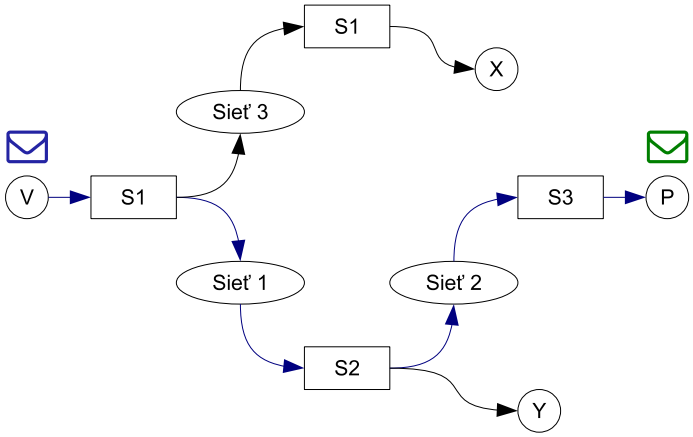
DATA - dáta TCP segmentu

4.3 Komunikácia na báze TCP/IP - sieťový pohľad



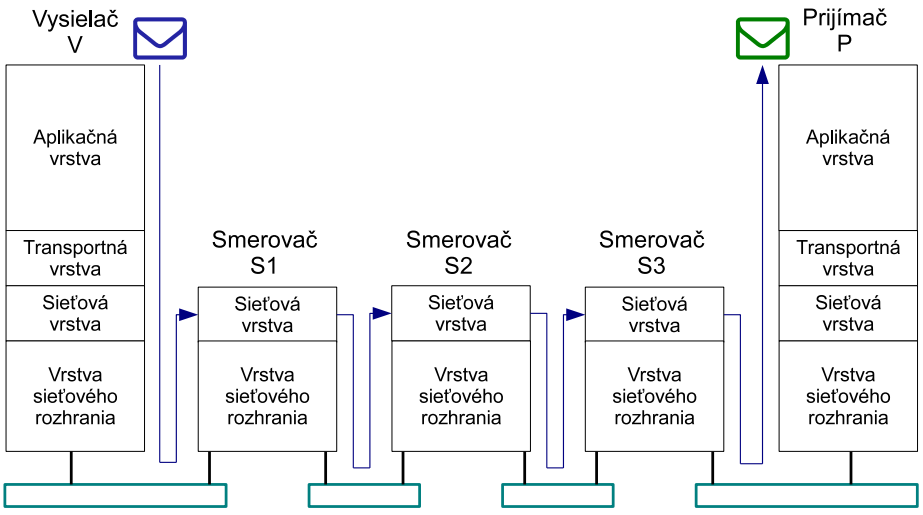
Obr. 6: Vzájomné porovnanie ISO-OSI modelu a TCP/IP modelu vo vrstvách

Na nasledujúcej schéme je znázornený prechod správy v sieti od vysielača po prijímač z pohľadu siete.



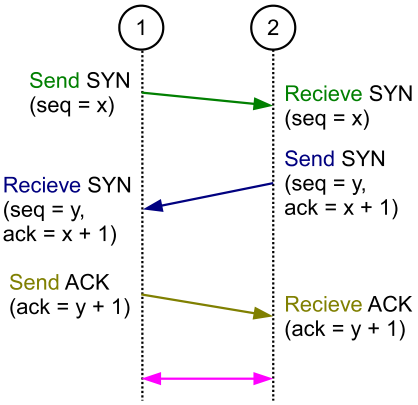
Obr. 7: Príklad posielania správy cez viacero sietí - sieťový pohľad

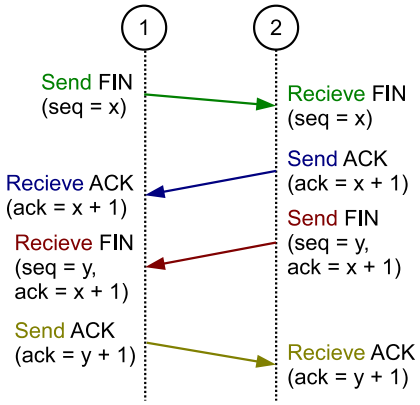
Na nasledujúcej schéme je znázornený rovnaký prechod správy v sieti od vysielača po prijímač z TCP/IP modelu.



Obr. 8: Príklad posielania správy cez viacero sietí - znázornené po vrstvách

4.4 Otváranie a zatváranie spojenia





4.5 Vzorový synchrónny TCP/IP klient

Toto sú najdôležitejšie časti kódu pre realizáciu synchrónneho TCP/IP klienta v programovacom jazyku C#. Triedy potrebné pre TCP/IP klient sú v

```
using System.Net;
using System.Net.Sockets;
```

Koncový bod resp. server je definovaný IP adresou a portom ako

```
IPEndPoint vzdialenyKB = new IPEndPoint(ServerIP, ServerPort);
```

Pred pokusom o pripojenie je nutné definovať TCP soket

```
Socket sender = new Socket(AddressFamily.InterNetwork, SocketType.Stream, ProtocolType.Tcp );
```

Nasleduje pripojenie klienta na soket - otvorenie spojenia. V tomto čase už server očakáva príjem

```
sender.Connect(vzdialenyKB);
```

Teraz môže klient pripraviť správu pre vysielanie. Túto správu je vhodné rozšíriť o koncový znak, ktorý ukončí vysielanie prípadne skladanie zložitejšej správy. Pri poslaní je možné získať informáciu o počte poslaných bytov.

```
byte[] buffer_vysielanie = Encoding.UTF8.GetBytes(sprava + KoncovyZnak);
int bajty_poslane = sender.Send(buffer_vysielanie);
```

V tomto prípade teraz server prijme správu, vykoná analýzu a spätne pošle potvrdzovaciu správu. Tentokrát je správa krátka a určite sa do buffera vojde. Taktiež je možné získať informáciu o počte prijatých bajtov.

```
int bajty_prijate = sender.Receive(buffer_prijem);
```

Nasleduje samozrejme úprava správy do požadovaného tvaru a jej výpis pre používateľa

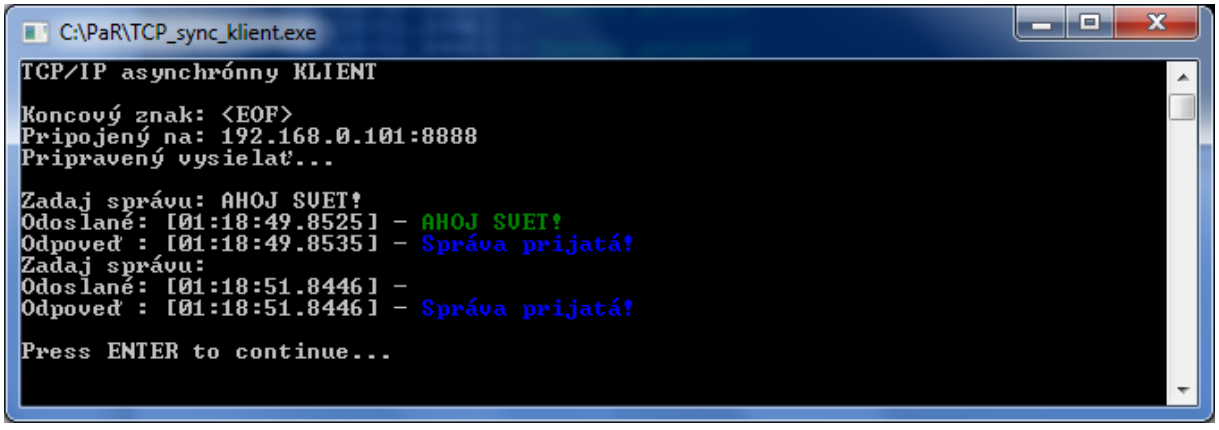
```
string sprava_prijata = Encoding.UTF8.GetString(buffer_prijem, 0, bajty_prijate);
                    Console.ForegroundColor = ConsoleColor.Blue;
                    Console.WriteLine(sprava_prijata);
```

Po skončení je možné zavrieť soket pre klient a server pomocou

```
sender.Shutdown(SocketShutdown.Both);
sender.Close();
```

Avšak je možné zachytiť tieto výnimky:

```
try
{ ... }
catch (ArgumentNullException ae){ Console.WriteLine("Chyba arg. - null: {0}",ae.ToString()); }
catch (SocketException se)     { Console.WriteLine("Chyba soketu: {0}",se.ToString()); }
catch (Exception e)        { Console.WriteLine("Neocakavana chyba: {0}", e.ToString()); }
```



Obr. 9: Príklad konzolového klienta pre TCP

4.6 Vzorový synchrónny TCP/IP server

Toto sú najdôležitejšie časti kódu pre realizáciu synchrónneho TCP/IP servera v programovacom jazyku C#. V prípade servera je nutné taktiež určiť adresu, avšak lokálneho pripojenia

```
IPEndPoint lokalnyKB = new IPEndPoint(LokalnaIP, ZvolenyPort);
```

Ďalej je rovnako nutné definovať soket pre TCP komunikáciu

```
Socket prijimac = new Socket(AddressFamily.InterNetwork, SocketType.Stream, ProtocolType.Tcp);
```

Medzi ďalšie kroky patrí naviazanie soketu na lokálny koncový bod a nastavenie fronty prichádzajúcich spojení(10)

```
prijimac.Bind(lokalnyKB);
prijimac.Listen(10);
```

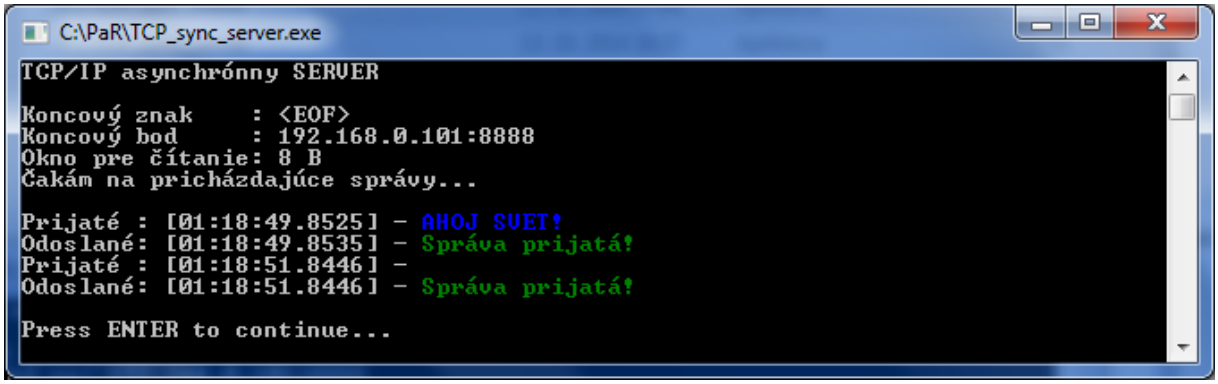
Nasleduje slučka, v ktorej program čaká na príchodzie správy od klienta na zvolenom sokete. Táto slučka môže skončiť v prípade detekcie dohodnutého koncového znaku. V tejto slučke

```
Socket soket = prijimac.Accept();
...
buffer_prijaty = new byte[VelkostBuffra];
int bajty_prijate = soket.Receive(buffer_prijaty);
sprava_prijata += Encoding.UTF8.GetString(buffer_prijaty, 0, bajty_prijate);
```

sa skladá text správy po menších častiach do výslednej správy. Po úspešnom prijatí celej správy server odpovedá klientovi potvrdzovacou správou napríklad

```
string sprava_potvrdenie = "Sprava prijata!";
byte[] buffer_potvrdenie = Encoding.UTF8.GetBytes(sprava_potvrdenie);
soket.Send(buffer_potvrdenie);
```

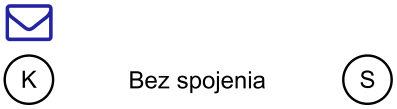
Rovnako ako klient, aj server vie uzatvoriť vzájomné spojenie.



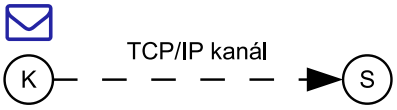
Obr. 10: Príklad konzolového servera pre TCP

4.7 Úloha pre riešenie na cvičení

Cieľom dnešného cvičenia je vytvorenie 2 aplikácii - klienta a server. Tieto aplikácie majú navzájom komunikovať pomocou TCP/IP protokolu a ich úlohou je navzájom si vymeniť 2 správy a to takto:

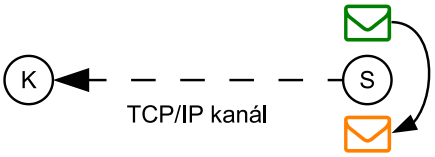


Krok 1:
K: obdrží správu pre vysielanie
S: čaká na otvorenie socketu

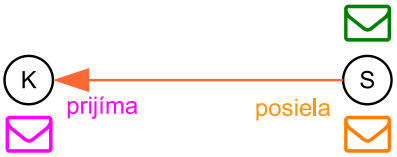


Krok 2:
K: otvorí TCP/IP kanál pomocou socketu so serverom a pripraví správu na prenos
S: čaká na správu od klienta na sockete

Krok 3:
K: posiela správu na socket
S: prijíma správu zo socketu a spracuje ju



Krok 4:
K: čaká na potvrdzovaciu správu zo servera
S: na základe prijatej správy vytvorí potvrdzovaciu správu



Krok 5:
K: prijme potvrdzovaciu správu zo servera
S: vyšle potvrdzovaciu správu klientovi

Krok 6:
K: spracuje správu a uzavrie socket
S: čaká na zatvorenie socketu

Úlohy pre riešenie na cvičení

- naprogramujte vlastný TCP klient ako WinForms aplikáciu (hlavná úloha)
- pošlite krátku správu (do 20 znakov) z klienta na server a späť (hlavná úloha)
- naprogramujte vlastný TCP server ako Konzolovú alebo WinForms aplikáciu (hlavná úloha)
- pošlite dlhú správu (nad 20 znakov) z klienta na server a späť krátku (rozšírenie)
- pošlite obsah súboru (viac ako 2000 znakov), ktorého názov je definovaný v textovom poli z klienta na server a uložte ho tam. Potvrďte v správe koľko bajtov bolo prenesených (bonusová úloha)

Spojenie medzi klientom a serverom ukončujte preddefinovaným koncovým znakom:

```
public static string KoncovyZnak = "<EOF>";
```

Riešte ako samostatné projekty v prostredí MS Visual Studio a celý Solution nahrajte v .ZIP forme do Moodle na konci cvičenia. V prípade nejasností kontaktujte cvičiaceho.