# Síťová komunikace

*Předmluva: síťová komunikace je spíše o kódu. Něco se dá sice říci o serveru, spíše ale natáhnete klienta, nicméně definice protokolů je většinou krátká a výmluvná. Přidejme si k tomu fakt že většinu z protokolů známe (aspoň by jsme měli) a zbývá nám jen kód. (Ideálně ho pochopit)*

Síťová komunikace v C# vyžaduje v namespace[**system.Net**](http://system.net)**.** Některé protokoly ještě **System.Net.Sockets.** V síťové komunikaci jsou nejméně 2 prvky, a to Server/Klient.

*>>doplňte otázkou 4.FTP Architektura Client-Server<<*

## Server

Naslouchá na určitém portu a IP adrese.

Přijímá dotazy od klienta, zpracovává je a vrací odpověď.

### dedikovaný

* vyhrazený pro speciální účely, bez přímého přístupu uživatelů

### nededikovaný

* server slouží uživateli zároveň jako obyčejný počítač

Hlavní rozdíl mezi osobním počítačem a serverem je ve vybavení programy (software). Současné operační systémy jsou obvykle univerzální a mohou sloužit jako osobní počítač i jako server. Rozdíl je pak v jejich nastavení, kdy u osobních počítačů je preferována interaktivita (počítač rychle reaguje na požadavky uživatele) a u serverů se klade důraz na škálovatelnost (schopnost dosažení co největšího výkonu).

## Klient

Komunikuje se serverem přes určitý port a zasílá dotaz.

Komunikovat můžeme pomocí několika protokolů.

Softwaroví klienti se hrubě člení na ***obsáhlé***, ***tenké*** a ***hybridní***. Vlastnosti rozhodující o kategorizaci vystihuje následující tabulka:

|  |  |  |
| --- | --- | --- |
|  | **Místní úložiště** | **Místní zpracování** |
| **Obsáhlý klient** | Ano | Ano |
| **Hybridní klient** | Ne | Ano |
| **Tenký klient** | Ne | Ne |

### Obsáhlý klient

Obsáhlý klient (anglicky *fat client*, *rich client* nebo *thick client*) provádí všechny požadované (v softwarovém návrhu vytyčené) operace sám — není závislý na konektivitě se serverem. Smyslem jeho spolupráce se serverem je synchronizace dat.

### Tenký klient

Tenký klient (*thin client*) využívá především systémové prostředky hostitelského počítače. Jeho agendou je víceméně jen prezentovat data poskytovaná aplikačním serverem, který vykonává převážnou část všech vyžadovaných operací. Výhodou takového řešení je vysoká ovladatelnost a pružnost. Jinými slovy, tenký klient nechá všechnu těžkou práci na serveru.

### Hybridní klient

Hybridní klient (*hybrid client*) je klient, který nesplňuje definice obsáhlého ani tenkého klienta — v některých klíčových operacích si vystačí sám, v jiných se neobejde bez interakce se serverem. Jako příklad lze uvést aplikaci nabízející čtení i zápis dat, která ze serveru načte statistický soubor a umožní nad ním provádět výpočty, aniž by tím vyvolávala požadavky na server, ale bez interakce se serverem tento soubor nedovolí rozšířit.

## UDP (User Diagram Protocol)

Jednoduchý protokol založený na posílání nezávislých zpráv. Nezachovává pořadí paketů. Prostě to “chrlí”. Když se jeden paket ztratí, nic se neděje.

Toto z něj dělá značně rychlejší protokol než je TCP.

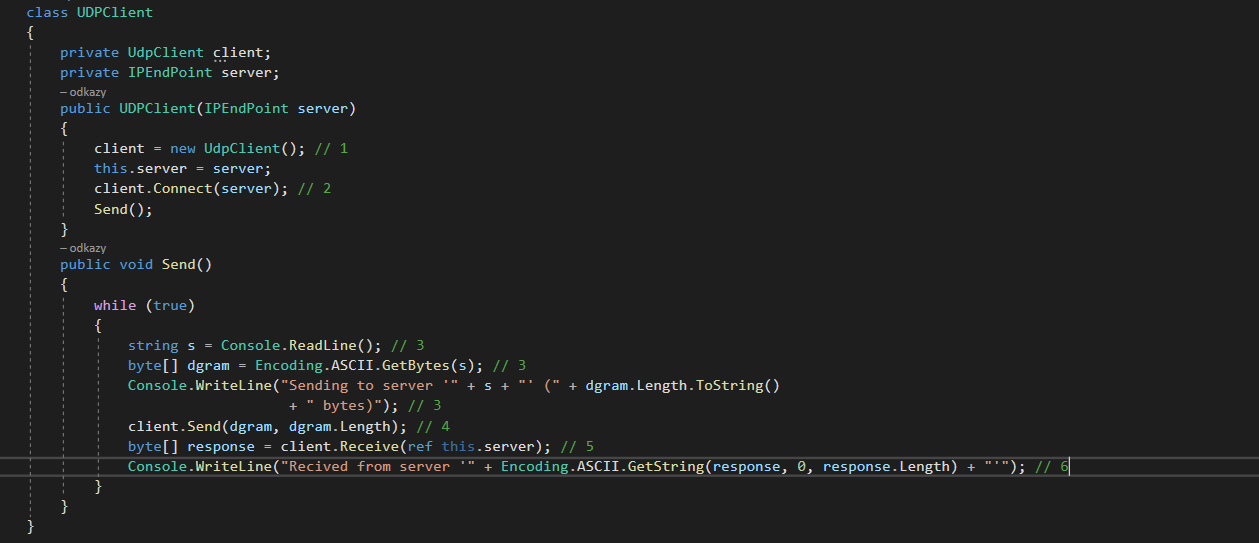
### Server

U UDP se server od klienta nijak zásadně nerozlišuje. Komunikace probíhá stylem odeslání paketů a očekáváním, že je na druhé straně někdo přijme.

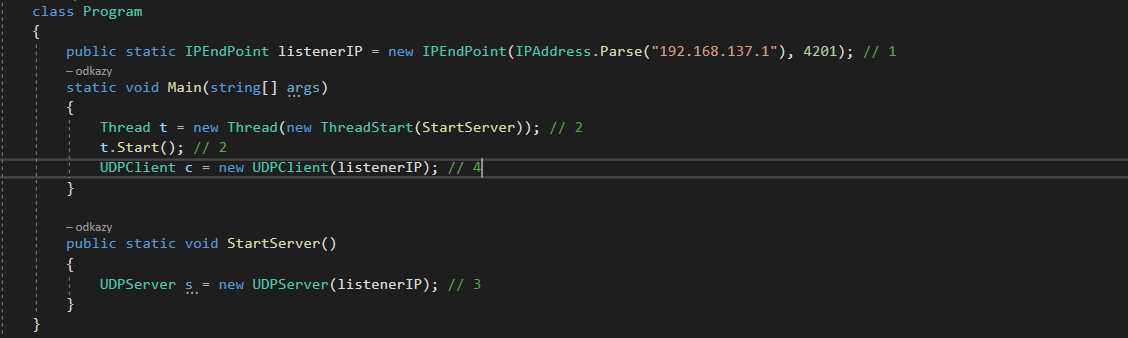
1. vytvoření nového UdpClienta (c# třída) na základě IP adresy a portu definovaných v konstruktoru
2. definování omezení IP adres a portů klientů pokoušejících se připojit. (např napsáním místo IPAddress.Any IPAdress.Parse(“192.168.2.5”) by program poslouchal pouze pakety přícházející z dané IP adresy)
3. samotné naslouchání (čekání až něco dorazí)
4. přetypování příchozích bytů na text a následné vypsání
5. vytvoření zprávy pro klienta a její přetypování na byty
6. odeslání vytvořených bytů zpět klientovi

### 

### Klient

1. vytvoření nového UdpClienta (c# třída). Vzhledem k tomu že jsme nezadali IP ani port. Vybere se prostě IP počítače na kterým klient běží a volný port
2. připojení klienta na náš server jehož IP jsme si poslali v konstruktoru. Tím klientovi v podstatě je říkáme kam má odesílat zprávy
3. načtení zprávy od uživatele z konzole, přetypování zprávy na byty a vypsání do konzole, že se pokoušíme něco odeslat
4. samotné poslání zprávy
5. očekávání odpovědi od serveru
6. přetypování a vypsání přijaté odpovědi

### Main

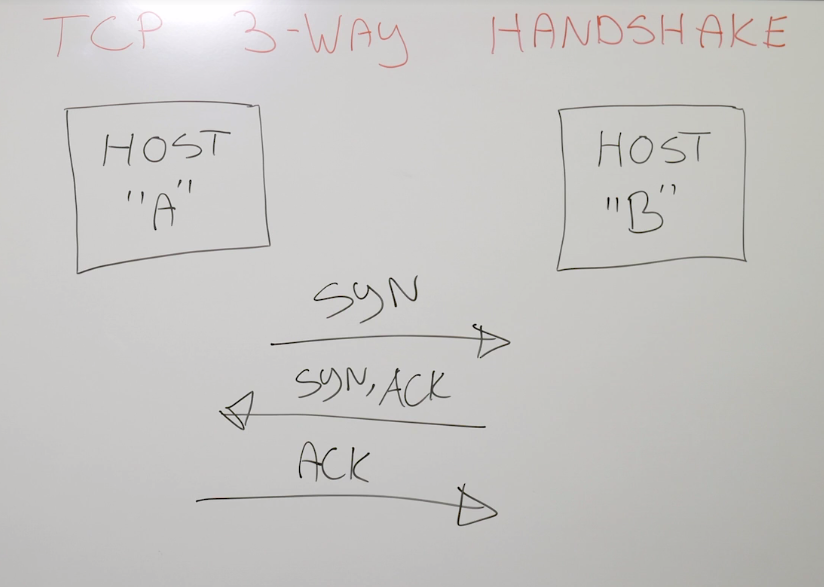
1. definování IP Adresy a portu pro server (lze použít např. IPAddress.Loopback, ale vyvarujte se IPAddress.Any)
2. vytvoření a spuštění vlákna s naším serverem
3. vytvoření našeho serveru
4. vytvoření našeho klienta (neplést s c# třídou UdpClient)

## TCP (Transmission Control Protocol)

Je spolehlivější než UDP. TCP je spojovaná transportní služba - musí před odesíláním dat navázat spojení mezi klientem a serverem - k tomu využívá tzv. Three way handshake. Tohoto protokolu využívá mnoho populárních aplikačních protokolů a aplikací na internetu, včetně WWW, e-mailu a SSH.

### Three way handshake

* Klient pošle požadavek - synchronizační paket (SYNCHRONIZE)
* Server vrátí vlastní sync. paket a potvrzení o tom že mu paket od klienta přišel (SYNC-ACKNOWLEDGE)
* Klient odpoví vlastním potvrzením o paketu (ACK)
* Spojení navázáno



### Server

1. vytvoří nový TcpListener na d portu zadaném v konstruktoru
2. spustí naslouchání
3. přijme nový pokus o navázání spojení - čeká se tu (blokující operace)
4. spustí nové vlánko pro klienta (klienta předává jako parametr)
5. přetypuje parametr (obecný object) na TcpClient (c# třída)
6. vytvoří pro klienta stream reader a writer - pomocí těchto dvou streamů s klientem komunikuje
7. čeká až mu klient pošle zprávu a tu pak vypíše
8. odepíše klientovi že mu zpráva přišla (sWriter.Flush() **nutný!**)

### 

### Klient

1. deklarujeme si nového TcpClienta (c# třída)
2. říkáme klientovi aby se připojil na server
3. vytvoříme StreamWriter a Reader pro komunikaci se serverem
4. necháme uživatele zadat zprávu
5. odešleme zprávu na server (sWriter.Flush() **nutný!**)
6. čeká na odpověď serveru a vypíše ji do konzole

### Main

V mainu pouze vytvoříme instance našeho serveru nebo klienta (oba nám na stejném pc fungovat nemohou, protože by využívali stejnou IP a port - lze otestovat na dvou počítačích, nebo server se dá otestovat přes putty)

## Univerzální řešení

### Server

1. Vytvořím si statickou proměnnou data typu byte a třídu socket.
2. Vytvořím si nový socket s typem socketu, jaký má protokol
3. Vytvořím si nový IPEndPoint a port.
4. Funkce Listen(); slouží k tomu, aby server věděl vlastně kolik může maximálně naslouchat uživatelů. V našem případě je to jen jeden.
5. Ani nemusím vysvětlovat, prostě si vytvořím nový socket s datama co přijdou.
6. Do proměnné typu byte se nám uloží data co přijdou.
7. Zjednodušeně řečeno : "přeuložím si data co přijdou do nové proměnné", abych nedekódoval stále přicházející data.
8. Dekóduji data pomocí metody Default, což podporuje tuším snad všechny znaky -> taky tam je možnost ASCII a tak podobně, no a pak jen tisknu data co přijdou na obrazovku.

### Klient

1. Zkusím se připojit na server. Pokud to půjde, dostaneme se dále. Pokud ne, smůla, vyskočí třeba nějaká chybová hláška.
2. Zkusím se tedy připojit pomocí funkce Connect. Naparsujeme si opět IP adresu ze stringu a zadáme port, který server naslouchá, tudíž 6666.
3. Uživatel zadá nějaký text, já si ho uložím a zakóduji pomocí metody default do proměnné, kterou jsem si vytvořil. Je to proměnná data typu byte.
4. Odešlu. 

## HTTP (Hypertext Transfer Protocol)

Postavený na protokolu TCP.

V tomto případě jsme většinou v roli klienta a komunikujeme se vzdáleným webserverem (nginx, apache). Využít můžeme například jednoduchou třídu WebClient. Ve většině případů posíláme 2 typy requestů, a to POST a GET.

### GET

* request pomocí url např. [https://www.youtube.com/*results***?search\_query=test+search**](https://www.youtube.com/results?search_query=test+search)
  + server uvidí search\_query jako jakousi proměnou se kterou může pracovat
  + fun fact: tyto “proměnné” v url ovlivňují SEO. Proto například u článků vidíte mnohdy v url název článku a ne pouze id článku.

1. deklarace WebClienta
2. stáhnutí HTML z url jako string
3. vypsání HTML

### POST

* odesílá uživatelská data na server
* používá se například při odesílání formuláře na webu
* s předaným objektem se pak zachází podobně jako při metodě GET. Data může odesílat i metoda GET (viz search\_query) , metoda POST se však používá pro příliš velký objem dat (víc než 512 bajtů, což je velikost požadavku GET), nebo pokud není vhodné přenášená data zobrazit jako součást [URL](https://cs.wikipedia.org/wiki/Uniform_Resource_Locator) (data předávaná metodou POST jsou obsažena v HTTP požadavku)

1. deklarace WebClienta
2. vytvoření listu pro naše proměnné, které budeme odesílat ( vyžaduje using System.Collections.Specialized)
3. definování našich proměnných
4. odeslání na server pomocí metody UploadValues() vracející HTML v bytech
5. vypsání na obrazovku

### Další typy requestů

(spíše něco navíc)

* **PUT**

Nahraje data na server. Objekt je jméno vytvořeného souboru. Používá se velmi zřídka, pro nahrávání dat na server se běžně používá FTP nebo SCP/SSH.

* **DELETE**

Smaže uvedený objekt ze serveru. K tomu je zapotřebí jistých oprávnění stejně jako u metody PUT.

* **TRACE**

Odešle kopii obdrženého požadavku zpět odesílateli, takže klient může zjistit, co na požadavku mění nebo přidávají servery, kterými požadavek prochází.

* **OPTIONS**

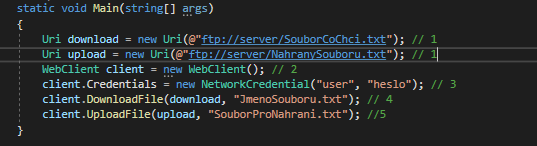
Dotaz na server, jaké podporuje metody.

* **HEAD**

Metoda podobná GET, avšak nepředává data. Poskytne pouze metadata o požadovaném cíli (velikost, typ, datum změny, …).

## FTP (File Transfer Protocol)

**FTP** je v informatice protokol pro přenos souborů mezi počítači pomocí počítačové sítě. Využívá protokol TCP. Využívá porty TCP/21 a TCP/20. Port 21 slouží k řízení a jsou jím také přenášeny příkazy FTP. Port 20 slouží k vlastnímu přenosu dat, který je 8bitový. Přenos může být *binární* nebo *ascii* (textový).V současné době už není považován za bezpečný a pomalu se nahrazuje za **SFTP**.

1. deklarace Uri (Url serveru + cesta k souboru) pro download a upload
2. deklarace WebClienta
3. nastavení ftp uživatele a hesla pro klienta
4. stažení souboru pomocí naší download uri (JmenoSouboru.txt odkazuje na lokální umístění pro stažený soubor)
5. nahrání souboru pomocí naší upload uri (SouborProNahrani.txt odkazuje na lokální umístění pro stažený soubor)

## SFTP (SSH File Transfer Protocol)

SFTP je protokol pro síťovou komunikaci, který pro svou funkci využívá SSH.

Používá se pro bezpečný přenos dat pomocí počítačové sítě. Obvykle se používá jako náhrada za jednoduchý protokol SCP. Jeho využití je velmi podobné normálnímu FTP.

**SSH** (**Secure Shell**) je v informatice označení pro program a zároveň pro zabezpečený komunikační protokol v počítačových sítích, které používají TCP/IP. SSH byl navržen jako náhrada za [telnet](https://cs.wikipedia.org/wiki/Telnet) a další nezabezpečené vzdálené shelly (rlogin, [rsh](https://cs.wikipedia.org/w/index.php?title=Rsh&action=edit&redlink=1) apod.), které posílají heslo v nezabezpečené formě a umožňují tak jeho odposlechnutí při přenosu pomocí počítačové sítě. Šifrování přenášených dat, které SSH poskytuje, slouží k zabezpečení dat při přenosu přes nedůvěryhodnou síť, jako je například Internet.