# 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)*

## 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. Skype

## 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. Nejprve si pořebujeme přesně definovat na které adrese a portu bude náš erver naslouchat
2. Vytvoříme si instance UDPClenta která se v podstatě nijak neliší od clienta, ale na rozdíl od clienta budeme infomace bindovat do instance tak aby server začal na zvolené kombinaci adrese a portu posloucha
3. Následně v cyklu budeme čekat než nám někdo něco pošle a v tu chvíli víme jeho ip addresu a port tak na tyto informace pošleme zpátky odpověď

import socket

localIP = "127.0.0.1"

localPort = 20001

bufferSize = 1024

msgFromServer = "Hello UDP Client"

bytesToSend = str.encode(msgFromServer)

# Create a datagram socket

UDPServerSocket = socket.socket(family=socket.AF\_INET, type=socket.SOCK\_DGRAM)

# Bind to address and ip

UDPServerSocket.bind((localIP, localPort))

print("UDP server up and listening")

# Listen for incoming datagrams

while(True):

bytesAddressPair = UDPServerSocket.recvfrom(bufferSize)

message = bytesAddressPair[0]

address = bytesAddressPair[1]

clientMsg = "Message from Client:{}".format(message)

clientIP = "Client IP Address:{}".format(address)

print(clientMsg)

print(clientIP)

# Sending a reply to client

UDPServerSocket.sendto(bytesToSend, address)

### Klient

1. Když vytváříme klienta musíme mít na mysli že cílem klienta je připojit se na server tím pádem musíme znát ip addresu a port server na který se budeme připojovat
2. Vytvoříme si instance UDPClenta která se v podstatě nijak neliší od serveru, ale na rozdíl od server nebudeme infomace bindovat do instance a využijeme je až ve chvíly kdy budeme posílat zprávu na server
3. String který posíláme musí být zakódován v bytech aby při přenosu nedošlo k chybám a aby byla zpráva rozkódovaná na druhé straně v pořádku
4. Násdně klient naslouchá podobně jako server než zas někdo pošle informace na kombinaci jeho adresy a portu

import socket

serverAddressPort = ("127.0.0.1", 20001)

bufferSize = 1024

# Create a UDP socket at client side

UDPClientSocket = socket.socket(family=socket.AF\_INET, type=socket.SOCK\_DGRAM)

# Load data from client

msgFromClient = input("Enter message:")

bytesToSend = str.encode(msgFromClient)

# Send to server using created UDP socket

UDPClientSocket.sendto(bytesToSend, serverAddressPort)

msgFromServer = UDPClientSocket.recvfrom(bufferSize)

msg = "Message from Server {}".format(msgFromServer[0])

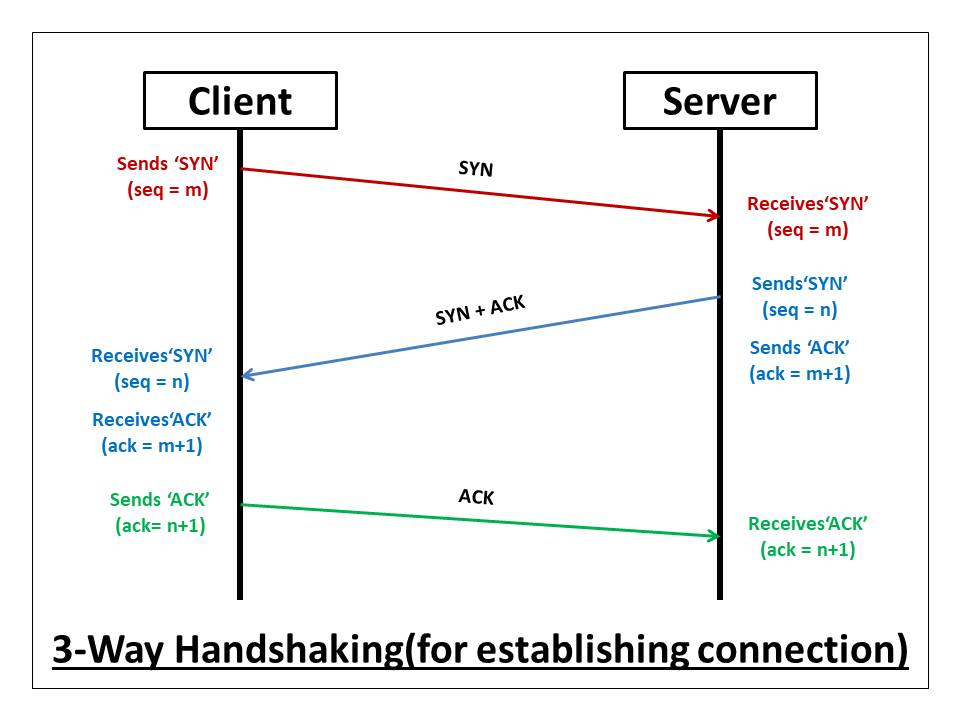
print(msg)

## 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 HTTP, SMTP 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

import socket,time

try:

server\_inet\_address = ("127.0.0.1", 65430)

server\_socket = socket.socket()

print("Server start on "+str(server\_inet\_address[0])+":"+str(server\_inet\_address[1]))

server\_socket.bind(server\_inet\_address)

server\_socket.listen()

connection, client\_inet\_address = server\_socket.accept()

try:

response=None

while True:

connection.send(bytes('user>','utf-8'))

response=connection.recv(1024).decode('utf-8').strip().replace('\r\n','')

#do something with client response

print(response)

time.sleep(0.01)

except:pass

finally:connection.close()

except:pass

finally:server\_socket.close()

### Server for more connection

import socket,time

import threading

def await\_user(connection):

try:

response=None

while True:

connection.send(bytes('user>','utf-8'))

response=connection.recv(1024).decode('utf-8').strip().replace('\r\n','')

#do something with client response

print(response)

time.sleep(0.01)

except:pass

finally:connection.close()

try:

server\_inet\_address = ("127.0.0.1", 65430)

server\_socket = socket.socket()

print("Server start on "+str(server\_inet\_address[0])+":"+str(server\_inet\_address[1]))

server\_socket.bind(server\_inet\_address)

server\_socket.listen()

connections =[]

while True:

print('Wait for the user')

connection, client\_inet\_address = server\_socket.accept()

connections.append((connection,

threading.Thread(target=await\_user,args=(connection,))))

connections[len(connections)-1][1].start()

except:pass

finally:server\_socket.close()

### Klient

import socket

host = '127.0.0.1'

port=65430

with socket.socket(socket.AF\_INET, socket.SOCK\_STREAM) as s:

s.connect((host, port))

while True:

data =s.recv(2048).decode()

while 'user' not in data:

data+=s.recv(2048).decode()

print(data,end='')

command = input()

s.send(bytes(command,'utf-8'))

## 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. pro ověření konektivity na stránku 'https://w3schools.com' pomocí get metody v pythonu
* v pythonu existuje modul request který se o http requesty stará

import requests

x = requests.get('https://w3schools.com')

print(x.status\_code)

### 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)

import requests

url = 'https://www.w3schools.com/python/demopage.php'

myobj = {'somekey': 'somevalue'}

x = requests.post(url, json = myobj)

print(x.text)

### 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. nastavení ftp uživatele a hesla pro klienta
3. stažení souboru pomocí naší download uri (JmenoSouboru.txt odkazuje na lokální umístění pro stažený soubor)

import ftplib

import sys

def getFile(ftp, filename):

try:

ftp.retrbinary("RETR " + filename ,open(filename, 'wb').write)

except:

print "Error"

ftp = ftplib.FTP("ftp.nluug.nl")

ftp.login("anonymous", "ftplib-example-1")

ftp.cwd('/pub/') #change directory to /pub/

file = getFile(ftp,'README.nluug')

ftp.quit()

## 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.

## SMTP port 25

je internetový protokol určený pro přenos zpráv elektronické pošty (e-mailů) mezi přepravci elektronické pošty (MTA). Protokol zajišťuje doručení pošty pomocí přímého spojení mezi odesílatelem a adresátem; zpráva je doručena do tzv. poštovní schránky adresáta, ke které potom může uživatel kdykoli přistupovat (vybírat zprávy) buď přímo na serveru, nebo z jiného počítače pomocí protokolů jako POP3 nebo IMAP. Jedná se o jednu z nejstarších aplikací, původní standard RFC 821 byl vydán v roce 1982 (v roce 2001 ho nahradil novější RFC 2821 s rozšířením ESMTP). SMTP funguje nad protokolem TCP, používá port TCP/25 pro komunikaci mezi poštovními servery a port TCP/587 pro příjem e-mailů od e-mailových klientů.

import smtplib

from email.mime.text import MIMEText

sender = 'admin@example.com'

receivers = ['info@example.com']

port = 1025

msg = MIMEText('This is test mail')

msg['Subject'] = 'Test mail'

msg['From'] = 'admin@example.com'

msg['To'] = 'info@example.com'

with smtplib.SMTP('localhost', port) as server:

# server.login('username', 'password')

server.sendmail(sender, receivers, msg.as\_string())

print("Successfully sent email")