

Sustav za mrežnu pohranu

Kolarević, Davor

Undergraduate thesis / Završni rad

2015

Degree Grantor / Ustanova koja je dodijelila akademski / stručni stupanj: **Josip Juraj Strossmayer University of Osijek, Department of Mathematics / Sveučilište Josipa Jurja Strossmayera u Osijeku, Odjel za matematiku**

Permanent link / Trajna poveznica: <https://um.nsk.hr/um:nbn:hr:126:225620>

Rights / Prava: [In copyright](#) / [Zaštićeno autorskim pravom.](#)

Download date / Datum preuzimanja: **2024-04-25**



Repository / Repozitorij:

[Repository of School of Applied Mathematics and Computer Science](#)



Sveučilište J.J. Strossmayera U Osijeku
Odjel za matematiku
Sveučilišni preddiplomski studij matematike

Davor Kolarević
Sustav za mrežnu pohranu

Završni rad

Osijek, 2015.

Sveučilište J.J. Strossmayera U Osijeku
Odjel za matematiku
Sveučilišni preddiplomski studij matematike

Davor Kolarević
Sustav za mrežnu pohranu

Završni rad

Voditelj: izv.prof.dr.sc. Domagoj Matijević

Osijek, 2015.

Sažetak

U ovom radu ćemo promatrati način kako se odvija komunikacija između servera i klijenta na internetu, te kako slati podatke. Za to su zaduženi internet protokoli kao što su HTTP i FTP koji su bazirani na nižem, transportnom TCP protokolu. Zbog rizika koji donosi slanje podataka putem interneta, SSL protokol kriptira podatke i osigurava sigurnu komunikaciju. Krajnji cilj je dizajnirati sustav za pohranu podataka koristeći TCP transportni protokol osiguran SSL-om.

Ključne riječi

Mreža, slanje podataka, server, klijent, protokol, kanal, mrežna vrata TCP, pasivni kanal, aktivni kanal, UDP, FTP, HTTP, SSL

Abstract

In this paper we will study methods of communications between server and client in the network and how to send data. This is a duty of internet protocols like HTTP and FTP that are based on lower, transport TCP protocol. Because of the risk that sending data over internet brings, SSL protocol encrypts data and secure safe communication. Final goal is to design system for cloud data storage using TCP transport protocol secured with SSL.

Key words

Network, sending data, server, client, protocol, socket, port, TCP, passive socket, active socket, UDP, FTP, HTTP, SSL

Sadržaj

1	Uvod	1
2	Mrežne aplikacije i kanali	2
3	Protokoli	3
3.1	TCP	3
3.1.1	TCP klijent	4
3.1.2	TCP server	5
3.1.3	Sigurnost	6
3.2	FTP	7
3.3	HTTP	8
4	O izradi projekta	10
5	Zaključak	14
6	Dodatak	15

1 Uvod

S pojavom računala i njihovom povećanom upotrebom u svakodnevnom životu, kako u privatnom i poslovnom dolazi do potrebe za pohranom podataka, tj. sigurnosnim kopijama u slučaju kvara računala. S tehnološkim razvojem povećavaju se i očekivanja prema pohranjivanju i čuvanju podataka. Podatci moraju biti dostupni bez obzira na vrijeme i mjesto, te moraju biti sigurno pohranjeni. Jedan od prvih medija koji se pojavio za pohranu je tzv. disketa (eng. Floppy Disk). Tanka pločica kvadratnog oblika unutar koje se nalazila okrugla magnetna pločica za pohranu mogla je pohraniti najviše par megabajta. Problem male memorije disketa riješen je pojavom kompaktnog diska, skraćeno CD (eng. Compact Disc) u kasnim 1970-ima. S kapacitetom od 700 MB mogao je pohraniti znatno veću količinu podataka od diskete. Ne tako davno pojavila se USB flash memorija (eng. USB flash drive) koja ima mogućnost višestrukog zapisivanja i brisanja podataka, te veću mehaničku otpornost od ostalih medija.

Zadnjih godina, sa sve većim razvojem interneta dolazi do pojave „mrežnog backupa“ ili mrežne sigurnosne pohrane. Sigurnosna kopija podataka čuva se na udaljenom računalu, serveru. Zbog toga korisnik u bilo kojem trenutku i sa bilo kojeg računala može pristupiti svojim podacima koji su spremljeni na serveru. Tvrtka koja posjeduje takve servere, klijentima za određenu cijenu nudi prostor za pohranu podataka. Pri tome, brigu o podacima koji su pohranjeni, te održavanje infrastrukture vodi poslužitelj servera. Za podatke kažemo da su pohranjeni u „oblaku“ (eng. Cloud storage) i na taj način su zaštićeni od mehaničkih kvarova kakvima su izloženi ostali mediji za pohranu. Upravo zbog toga je mrežni backup postao jedan od glavnih načina spremanja podataka. S druge strane, potencijalni problem je sigurnost podataka pošto se komunikacija odvija putem interneta. Postoji mogućnost da će prilikom prijenosa ili dok su pohranjeni na serveru, podatke pročitati netko kome nisu namijenjeni.

Danas Microsoft, Google, i drugi servisi kao Dropbox nude korisnicima velika količine prostora za spremanje podataka te dosadašnji način spremanja podataka koristeći CD-e, DVD-e i USB memoriju čine gotovo nepotrebnim.

U prvom poglavlju objašnjena je arhitektura mrežnih aplikacija, značenje servera i klijenta te kako se poruke šalju kroz vrata (engl. Socket). Drugi dio razrađuje glavne protokole zadužene za slanje podataka (HTTP i FTP) i transportne protokole (TCP i UDP). Opisana je i jednostavna implementacija TCP servera i klijenta. Zbog korištenja SSL protokola u praktičnom dijelu rada obuhvaćena je i važnost sigurne komunikacije preko interneta te kako se ona postiže. Cilj zadnjeg poglavlja je korištenje stečenog znanja i implementiranje sustava za mrežnu pohranu koristeći TCP protokol. Program se sastoji od serverskoj i klijentskog dijela koji se poziva upisom ključne riječi server ili client u naredbenom retku.

2 Mrežne aplikacije i kanali

Cilj stvaranja mrežne aplikacije je napisati program koji će raditi na različitim računalima i komunicirati međusobno s drugima preko interneta. U klijentsko-serverskoj arhitekturi tu je poslužitelj, server, koji je uvijek dostupan korisnicima, koji se zovu klijenti. Npr. korisnik preko web preglednika (eng. browser) sa nekog uređaja (laptop, tablet, mobitel itd.) šalje zahtjev web serveru za određenom web stranicom. Kada server primi zahtjev šalje stranicu nazad klijentu. Server uvijek ima stalnu, poznatu adresu zvanu IP adresa, tako da klijent u svakom trenutku može poslati zahtjev na adresu servera. Kod klijentsko-serverske arhitekture bitno je da klijenti nikada ne komuniciraju međusobno, nego uvijek preko servera. Mrežna aplikacija se sastoji od parova procesa – programa koji su zaduženi za komunikaciju. Proces na jednom kraju kreira poruku i šalje ju procesu primatelju koji po potrebi šalje odgovor. Procesi šalju i primaju poruke kroz sučelje zvano kanal (engl. socket).

Kanal najlakše možemo shvatiti kao vrata kroz koji prolaze poruke pri međusobnoj komunikaciji između procesa. Kanal je sučelje koje povezuje aplikacijski i transportni sloj. Kako bi poruka mogla biti poslana na određeno odredište proces koji prima poruku mora imati adresu. Na internetu svako je računalo identificirano IP adresom, što je 32-bitni broj. Kada poruka stigne, proces koji šalje poruku mora odrediti na koji proces mora proslijediti poruku, tj. kroz koji kanal mora proći. Ta je informacija potrebna zbog toga što računalo na koje je poruka stigla može imati pokrenuto više mrežnih aplikacija. U tu svrhu su uvedena mrežna vrata(engl. Port).

Kako je već spomenuto, da bi proces poslao poruku, mora ju prvo poslati kroz kanal. Na drugoj strani kanala, transportni protokol je zadužen za prijenos poruke do kanala koji treba primiti poruku. Internet nudi više protokola za komunikacija, a na dizajneru aplikacije je da izabere onaj koji najbolje odgovara njegovoj aplikaciji.

3 Protokoli

Mrežni protokol definira pravila za komunikaciju između mrežnih uređaja. Pomoću posebnih mehanizama omogućuju im da se prepoznaju na mreži i uspostave konekciju kako bi mogli razmjenjivati podatke.

Dva najbitnija transportna protokola su UDP (User Datagram Protocol) i TCP (Transmission Control Protocol). Kada programer krene sa razvojem svoje mrežne aplikacije, jedna od prvih odluka je odabir jednog od ova dva protokola. TCP nudi pouzdan prijenos podataka. Drugim riječima, TCP protokol nudi garanciju da su svi podatci koju su trebali biti poslani stvarno poslani. S druge strane, UDP ne nudi tu garanciju, te poruke koje su primljene ne moraju biti istim redoslijedom kako su i poslone. Zbog toga je UDP prigodan za aplikacije kojima je bitnija brzina od gubitka podataka, npr. Internet telefon, televizija i mrežne igre. U Pythonu, TCP i UDP kanali dio su Pythonove standardne biblioteke *socket*. Da bi napravili novi TCP kanal koristit ćemo naredbu:

```
socket.socket(socket.AF_INET, socket.SOCK_STREAM),
```

a da bi napravili UDP kanal:

```
socket.socket(socket.AF_INET, socket.SOCK_DGRAM).
```

Oznaka *socket.AF_INET* označava da je riječ o IPv4 familiji adresa, *socket.SOCK_STREAM* označava TCP, a *socket.SOCK_DGRAM* UDP transportni protokol.

Protokoli aplikacijskog sloja definiraju kako procesi mrežne aplikacije, koji rade na različitim računalima, međusobno komuniciraju, odnosno međusobno prosljeđuju poruke. Aplikacijski protokoli definiraju tip poruke koju treba poslati ili koja je primljena (zahtjev ili odgovor), sintaksu i pravila kako bi program lakše mogao utvrditi kada i kako proces treba poslati poruku ili odgovor.

3.1 TCP

Kao što je već spomenuto, TCP (Transmission Control Protocol) je transportni protokol koji garantira da su sve poruke koje su trebale biti poslone stvarno i poslone u točnom redoslijedu. Protokoli koji služe za razmijenu datoteka i dokumenata, kao što su HTTP i FTP, gotovo su uvijek bazirani na TCP-u. Prije nego što klijent i server počnu međusobno razmjenjivati poruke, prvo moraju uspostaviti TCP konekciju. Server čeka klijenta da prvi pošalje zahtjev,

te mora uvijek biti dostupan i spreman na konekciju. Zbog toga, server mora imati otvoren poseban kanal, pasivni kanal, koji ima zadatak primati konekcije od strane klijenata i pamti njihovu adresu i mrežna vrata(engl. Port). Primanje ili slanje podataka se ne odvija preko pasivnog kanala. Kada klijent pošalje zahtjev, pasivni kanal stvara novi kanal, aktivni kanal, koji je dodijeljen isključivo tom klijentu preko kojeg će razmjenjivati podatke. Ovaj način uspostave konekcije između servera i klijenta naziva se „handshake“ postupak, te je on potpuno nevidljiv klijentskom i serverskom programu. Prema [2, str. 164.-167.], u programskom jeziku Python, implementacija jednostavnog TCP servera i klijenta bi izgledala na sljedeći način.

3.1.1 TCP klijent

```
import socket

imeServera = "servername"
port = 999
klijentski_socket = socket.socket(socket.AF_INET, \
                                   socket.SOCK_STREAM)
klijentski_socket.connect((imeServera, port))
poruka = "pozdrav serveru"
klijentski_socket.sendall(poruka)
odgovor = klijentski_socket.recv(1024)
print(odgovor)
klijentski_socket.close()
```

Prva stvar koju treba napraviti je uključiti biblioteku *socket* koja sadrži osnovne naredbe za stvaranje mrežnih aplikacija. Da bi se klijent mogao spojiti na server mora znati adresu i vrata servera. Linija

klijentski_socket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)

koda stvara klijentski socket. Parametar `socket.AF_INET` označava da program koristi IPv4 za konekciju, a `socket.SOCK_STREAM` da je riječ o TCP kanalu. Klijent uspostavlja konekciju sa serverom pomoću metode `connect()`, a parametar koji joj treba proslijediti je uređen par adrese i vrata servera.

klijentski_socket.sendall(poruka)

Metoda `sendall` šalje podatke na server s kojim je uspostavljena konekcija i omogućava da svi paketi koji se šalju budu i poslani. Kada server pošalje poruku klijentu, ona se sprema u varijablu *odgovor*. Linija *klijentski_socket.close()* zatvara klijentski kanal i TCP konekciju sa serverom.

3.1.2 TCP server

```
import socket

serverPort = 999
serverSocket = socket.socket(socket.AF_INET, \
                             socket.SOCK_STREAM)
serverSocket.bind(('', serverPort))
serverSocket.listen(1)
while True:
    connectionSocket, adresa = serverSocket.accept()
    poruka = connectionSocket.recv(1024)
    odgovor = poruka.upper()
    connectionSocket.sendall(odgovor)
    connectionSocket.close()
```

Kao kod klijenta stvara se novi TCP kanal i naredbom *bind()* veže se za određenu adresu i vrata. Tako kada god netko pošalje poruku na adresu i vrata koja je dodijeljena serveru, ona će biti poslana kroz ovaj kanal. Kanal koji je kreiran biti će pasivni kanal koji će čekati na klijenta da pošalje zahtjev za uspostavom konekcije.

pasivniSocket.listen(1)

Server čeka na TCP konekciju, a broj koji treba proslijediti naredbi *listen()* je maksimalan broj veza na čekanju. Kako je već spomenuto, server uvijek mora biti dostupan da primi zahtjev od klijenta. Zbog toga je potrebna beskonačna petlja.

aktivniSocket, adresa = pasivniSocket.accept()

Kada klijent pošalje zahtjev za uspostavom konekcije, naredba *accept()* za *pasivniSocket* vraća uređen par novog kanala, koji će biti zadužen za primanje i slanje podataka (dakle, aktivni kanal), i adrese kanala koji je vezan za klijenta. Ovime je završen proces uspostave

veze između klijenta i servera, točnije između `aktivniSocket` i `klijentski_socket` (iz gornjeg koda implementacije klijenta).

`aktivniSocket.close()`

Nakon što server odgovori klijentu i pošalje mu izmijenjenu poruku, naredbom `close()` zatvara se `aktivniSocket`, ali `pasivniSocket` i dalje ostaje otvoren i čeka na novog klijenta da se spoji.

3.1.3 Sigurnost

Većina mrežne komunikacije se odvija preko TCP protokola, no on ne nudi nikakvu sigurnost pri razmijeni podataka. Svi podatci koji se šalju putuju cijelom mrežom točno onakvi kakvi oni stvarno jesu, bez enkripcije, pa bilo tko može presresti našu poruku i pročitati njen sadržaj. Zbog toga je razvijen SSL (engl. Secure Socket Layer), poboljšanje u smislu sigurnosti za TCP. SSL sve podatke koji se šalju preko mreže prvo šifrira, tako da u slučaju presretanja poruke od treće strane, svi podatci budu zaštićeni. Kako bi se postigla SSL zaštita, u programski kod treba biti uključen SSL kod, i na klijentskoj i na serverskoj strani programa.

SSL koristi tehniku javnog i privatnog ključa koja radi na sljedeći način: Javni ključ se koristi sa šifriranje poruku i smije biti vidljiv svakome jer s njime nije moguće dešifrirati poruku. Poruku je jedino moguće dešifrirati koristeći privatni ključ koji treba ostati tajan između onoga tko šalje poruku i kome je namijenjena.

Osim važnosti šifriranja poruka, bitan je i *integritet poruke* (engl. Message integrity), tj. da klijent zna da je poruka koju je dobio stvarno poslao server. Bilo tko se može ubaciti u komunikaciju između servera i klijenta i ma da ne razumije poruku, može promijeniti njezin sadržaj. Kako bi se to izbjeglo, koristi se provjera vjerodostojnosti poruka – MAC (engl. Message Authentication Code). Šifriranoj poruci dodaje se (konkretira) MAC ključ tako da klijent može utvrditi je li primljena poruka stvarno poslana od servera, i obrnuto.

Pretpostavimo da klijent želi uspostaviti vezu sa serverom koristeći TCP vezu poboljšanu SSL-om. Prema [2, str.713], nakon što klijent pošalje zahtjev za spajanje na server (tzv. „hello poruka“), komunikacija se odvija na sljedeći način:

1. Klijent šalje listu kriptografskih algoritama koje podržava.
2. Server odabire algoritam i šalje svoj odabir natrag klijentu zajedno sa javnim ključem i certifikatom koji potvrđuje klijentu da je sigurno uspostaviti vezu.

3. Klijent kreira nasumični broj PMS (engl. Pre-Master Secret), šifrira ga pomoću javnog ključa koji je dobio od servera i šalje na serveru.
4. Server dešifrira PMS pomoću privatnog ključa te pomoću njega, server i klijent nezavisno jedan o drugome stvaraju MS (engl. Master Secret). Pomoću MS-a, oboje generiraju po 4 ključa:
 - E_s - ključ za šifriranje poruka koje šalje server (njime klijent dešifrira poruke)
 - E_k - ključ za šifriranje poruka koje šalje klijent (njime server dešifrira poruke)
 - M_s - MAC ključ za podatke koje šalje server
 - M_k - MAC ključ za podatke koje šalje klijent

Poruke se ne šalju u cijelosti. SSL dijeli poruku u više manjih dijelova odgovarajućih duljina. Svakom djeliću (dalje označenom slovom d) konkatencijom se pridodaje MAC ključ M . Dobiva se novi zapis, $d+M$, te se zatim na njega djeluje hash funkcijom. Hash funkcija zapis proizvoljne duljine pretvara u hash vrijednost fiksne duljine. Prije slanja zapisa, potrebno ga je šifrirati ključem E . Šifrirani zapis se zatim šalje TCP vezom.

3.2 FTP

FTP (eng. File Transfer Protocol) je protokol za razmjenu datoteka putem interneta između dva računala. Kako bi klijent mogao pristupiti udaljenom računalu sa kojega želi preuzeti ili primiti datoteku, mora upisati korisničko ime i lozinku. FTP za prijenos podataka koristi TCP vezu, točnije dvije TCP veze: nadzornu i podatkovnu. Nadzorna veza se koristi za slanje informacija o korisničkom imenu i lozinki klijenta, naredbama za promjenu direktorija te naredbi kao što su put (slanje datoteka na server) i get (preuzimanje datoteka sa servera). Druga veza, podatkovna, služi za prijenos podataka.

Kada klijent započne FTP vezu sa serverom, prvo postavlja kontrolnu TCP vezu sa serveru na vratima 21. Preko kontrolne veze klijent serveru šalje korisničko ime i lozinku. Kada server primi naredbu za prijenos podataka, server tada postavlja podatkovnu TCP vezu sa klijentom. FTP tada šalje točno jednu datoteku i nakon toga zatvara podatkovnu vezu. Ukoliko tijekom prijenosa datoteke između servera i klijenta, klijent želi prenijeti još jednu datoteku, FTP otvara još jednu podatkovnu vezu. Prema tome, kontrolna veza ostaje otvorena cijelo vrijeme, dok se nova podatkovna veza otvara za svaku datoteku koja se razmjenjuje.

Tokom cijele veze sa klijentom, server mora pratiti njegovo stanje. Točnije, mora znati koji serverski direktorij klijent ima otvoren. Takav način rada prilično ograničava broj veza koje FTP može uspostaviti.

3.3 HTTP

HTTP (eng. Hyper Text Transfer Protocol) je protokol za prijenos podataka baziran na TCP transportnom protokolu te je glavni protokol za prijenos informacija na Webu. HTTP je implementiran u dva programa: klijentskom i serverskom. Klijentski i serverski program komuniciraju preko mreže izmjenjujući HTTP poruke. HTTP definira strukturu tih poruka te kako se one izmjenjuju. U kontekstu pristupanja sadržaja na Webu, Web preglednici (Internet Explorer, Google Chrome, Firefox) implementiraju klijentsku stranu HTTP-a. Web serveri na kojima se nalaze Web stranice i drugi objekti kojima možemo pristupiti preko URL adrese, čine serversku stranu HTTP-a. Kada korisnik zatraži Web stranicu, Web preglednik pošalje HTTP zahtjev Web serveru. Kada server primi zahtjev, HTTP šalje odgovor u kojemu se nalaze objekti koji čine Web stranicu koju je klijent zatražio. HTTP koristi TCP kao osnovni transportni protokol. Prema tome, TCP pruža HTTP-u pouzdan prijenos podataka.

Za razliku od FTP-a, HTTP ne pamti stanje o klijentu koji se spojio na server. To omogućava dizajniranje Web servera koji podržavaju tisuće istovremenih TCP konekcija. No ponekad je poželjno da Web stranica identificira klijenta, bilo da zabrani određeni sadržaj klijentu ili da mu omogući prijavu na Web stranicu. U tu svrhu uvedeni su tzv. Kolačići (eng. Cookies). Kada klijent pošalje zahtjev Web serveru, server stvara jedinstveni identifikacijski broj koji šalje nazad klijentu. Kada Web preglednik primi odgovor od servera, sprema identifikacijski broj koji je dobio od servera u posebnu datoteku. Tako svaki put kada klijent zatraži novu Web stranicu od servera, Web preglednik zajedno sa zahtjevom šalje identifikacijski broj koji je dobio od servera te na taj način server može pratiti aktivnost svakog klijenta na Web stranici.

Najčešći način na koji klijent dohvaća sadržaj sa servera je metodom GET.¹ Prema [1, str. 143] tipični HTML zahtjev izgleda na sljedeći način:

¹osim metode GET, u slučaju popunjavanja formi (pretraživanje, ispunjavanja obrazaca) koristi se metoda POST, koja isto tako vraća traženi sadržaj, ali prilagođen s obzirom što je korisnik upisao u formu

GET /rfc/rfc2616.txt HTTP/1.1

Host: ww.ietf.org

Prvi redak sadrži informaciju o metodi i dokumentu koji klijent zahtjeva, a drugom je upisana adresa poslužitelja. Razlog zbog kojega je HTTP bolje rješenje od FTP protokola je što HTTP ne mora održavati stalnu vezu sa klijentom zbog čega može posluživati više klijenata istovremeno. Posluživanje klijenata dodatno ubrzava proxy poslužitelj koji djeluje između klijenta i servera. Proxy poslužitelj u svojoj memoriji (engl. Web cache) čuva kopije od nedavno zatraženih sadržaja. Prema [2, str. 110] Proxy server radi na sljedeći način: 1. Klijent uspostavlja TCP vezu sa proxy serverom i šalje HTTP zahtjev za datotekom. 2. Proxy server provjerava ima li datoteku spremljenu u lokalnoj memoriji. Ako ima, šalje HTTP odgovor klijentu sa datotekom koju je zatražio. 3. Ako nema, Proxy server uspostavlja TCP vezu sa serverom na koje se datoteka originalno nalazi, šalje HTTP zahtjev i sprema datoteku u svoju lokalnu memoriju. 4. Proxy server šalje datoteku klijentu, te ako nakon njega drugi klijent zatraži isti sadržaj, može mu ga odmah poslati bez da uspostavlja vezu sa serverom gdje se originalno nalazi. Prema tome, Proxy server se ponaša kao server, ali i kako klijent. Proxy server je uobičajeno instaliran od strane pružatelja internetska usluge².

²ISP – Internet Service Provider

4 O izradi projekta

Cilj ovog projekta je bio napraviti sustav za razmjenu podataka između servera i klijenta koristeći TCP protokol gdje je bilo potrebno napraviti vlastiti protokol za razmjenu datoteka. Program se sastoji od serverskog i klijentskog dijela program podijeljenog u dvije klase: *Backup_Server* i *Backup_Client*. Server u posebnoj datoteci *backup_server.txt* čuva podatke o adresi i vratima servera te veličini memorije (u MB) za spremanje podataka koja se dodjeljuje svakom korisniku, dok klijent u *backup_client.txt* čuva podatke o putanji direktorija koji treba poslati na server, adresu i vrata servera te vremenski interval slanja podataka na server (u sekundama). Pri pokretanju programa klijent odabire želi li pokrenuti serverski ili klijentski program. Klijent šalje datoteke iz odabranog direktorija na server, pri čemu program vodi računa o tome ukoliko je došlo do neke izmjene datoteka unutar odabranog direktorija. U tom slučaju program izmijenjenu datoteku ili datoteke, ako je došlo do više promjena šalje ponovno na server, ali na taj način da napravi novu datoteku na strani servera. Dio klijentskog programa koji se brine o vremenu promjena datoteka izvršava se u posebnoj programskoj niti *tstamp* i radi neovisno o glavnom dijelu programa.

Koristeći metodu *socket.socket(socket.AF_INET, socket.SOCK_STREAM)* iz biblioteke *Socket* kreiran je TCP kanal na strani servera i klijenta za međusobnu komunikaciju. Kako bi se osigurala sigurna komunikacija, kanal je zaštićen SSL-om. Iz biblioteke *SSL* koristi se metoda *wrap_socket()* koja postojeći TCP kanal poboljšava SSL zaštitom u smislu da svi podaci koji se šalju budu kriptirani. Na serverskoj strani metodi *wrap_socket()* treba proslijediti i javni ključ *server.key* koji server šalje klijentu, te certifikat *server.crt* kako bi server potvrdio klijent da je sigurno spojiti se na njega.

```
#klijent
```

```
sock = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
print("Spajam se na server na %s, port: %d" \
      %(self.remote_host, self.remote_port))
try:
    sock.connect((self.remote_host, self.remote_port))
except:
    print("Spajanje sa serverom nije uspjelo...")
    sys.exit(1)
self.s=ssl.wrap_socket(sock, ssl_version=ssl.PROTOCOL_SSLv3)
```

```
#server
```

```
self.s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
self.s.setsockopt(socket.SOL_SOCKET, socket.SO_REUSEADDR, 1)
self.s.bind((self.adresa, self.port))
print("Otvaram pasivni socket na ", self.s.getsockname())
self.s.listen(1)
```

```

# primanje novih klijenata
while True:
    try:
        (sock, client_adresa) = self.s.accept()
    except:
        print("Otpustam klijenta...")
        sock.close()
        continue

client_socket = ssl.wrap_socket(sock, server_side=True, \
                                ssl_version=ssl.PROTOCOL_SSLv23, \
                                cert_reqs=ssl.CERT_NONE, keyfile="server.key", \
                                certfile="server.crt")

```

Serverski program čeka da se klijent spoji i unese ime preko kojeg će ga server raspoznavati i istovremeno kreirati direktorij s njegovim imenom.

```

# dokumentiranje klijenta
nick = client_socket.recv(20).decode()
if nick == "":
    nick = "Anonymous"
self.clients[client_socket] = [nick, self.quota]
mapa = os.getcwd()+"\\"+self.clients[client_socket][0]
if not os.path.isdir(mapa):
    os.makedirs(mapa)

```

Kako bi server mogao posluživati više klijenata istovremeno, metodom *Thread()* iz Threading biblioteke stvara se nova nit za svakog klijenta. Na strani klijent program prvo metodom *os.walk()* izlista sve datoteke u direktoriju i poddirektorijima. Program šalje datoteku na taj način da prvo pošalje veličinu datoteke koju treba poslati na server i njezino ime kako bi se na strani servera mogla otvoriti prazna datoteka s istim imenom.

Na serverskoj strani, metoda *receive_files()* zadužena je za primanje datoteka koje šalje klijent. Server otvara praznu datoteku metodom *open()*, na ime koje je primio od klijenta. Kako je moguće da je datoteka dio poddirektorija u klijentovoj mapi, potrebno je otvoriti isti poddirektorij i na strani servera te u njega valjano spremiti datoteku. Za to je zadužen sljedeći dio metode, koji na osnovi primljenog imena provjera da li je datoteka dio poddirektorija (s obzirom na element „
“ u imenu datoteku).


```

if ime.count("\\") != 0:
    Lista=[ime.rsplit("\\",j)[0] for j in range (1,ime.count(
        "\\")+1)]
    for i in range(1,ime.count("\\")+1):
        mapa = os.getcwd()+"\\"+ \
            self.clients[client_socket][0]+"\\"+Lista[-i]
        if not os.path.isdir(mapa):
            os.makedirs(mapa)
        ime_datoteke = mapa+"\\"+ime.rsplit("\\",1)[1]
else:
    ime_datoteke = self.clients[client_socket][0]+"\\"+ime

```

Kako program dozvoljava čuvanje starih stanja datoteka, u slučaju njenog ponovnog slanja na server, potrebno je provjeriti postoji li već datoteka na strani servera. Za to je zadužena metoda *save_file()* koja rekurzivno provjerava postojanje kopija na serveru. Broj kopija ograničen je na 10.

```

def save_file(self,filename,broj):
    if broj > 10:
        return ""
    novo = "%s(%d).%s" %(filename.rsplit(".",1)[0], \
        broj,filename.rsplit(".",1)[1])
    if os.path.exists(novo):
        return self.save_file(filename,broj+1)
    else:
        return novo

```

Primanje se odvija po dijelovima (chunkovima) tako da server vodi računa o tome jesu li pristigli svi paketi pošto poznaje veličinu datoteke koju prima.

```

while True:
    datoteka.write(chunk)
    chunk = client_socket.recv(1024)
    velicina_datoteke = velicina_datoteke - len(chunk)
    #print(velicina_datoteke)
    if velicina_datoteke == 0:
        datoteka.write(chunk)
        break

```

Nužno je napomenuti da prije svakog primanja server prvo provjerava ima li klijent dovoljno memorije u svome direktoriju na strani servera. U slučaju premašene dozvoljene kvote server prekida slanje. O tome jesu li datoteke ažurne i je li potrebno ponovno slanje datoteka na server brine se posebna programska nit na strani klijenta. Metoda *timestamp()*, koja se cijelo vrijeme vrti u pozadini klijentskog programa, u zadanim vremenskim intervali prolazim direktorijem i provjerava je li došlo do promjena određenih datoteka. Provjeru vrši metoda *getmtime()* iz Pythonove biblioteke *os*, koja vraća vrijeme zadnje promjene datoteke. Cjelokupni programski kod programa za mrežnu pohranu *Backup.py*, dostupan je u dodatku.

5 Zaključak

Zbog složenosti slanja velikih datoteka, koristiti TCP protokol za stvaranje vlastitog protokola nije najjednostavnije, pogotovo što za tu svrhu postoje već pouzdani podatkovni protokoli kao HTTP i FTP. HTTP, se pokazuje kao najbolji izbor budući da može istovremeno održavati konekciju s znatno više klijenata nego što to može nešto stariji FTP protokol. TCP zbog mogućnosti da svi poslani podatci stvarno stignu na odredište pokazuje se kao bolji izbor za slanje datoteka koje su osjetljive na gubitak podataka. Pri tome, ovisno o namijeni aplikacije, treba voditi računa o sigurnosti prijenosa podataka i uključivanja SSL protokola u aplikaciju.

6 Dodatak

```
1 import socket
2 import sys
3 import threading
4 import os.path
5 import pickle
6 import time
7 import ssl
8
9
10 class BackupServer:
11     def __init__(self,adresa,port,quota):
12         self.adresa = adresa # adresa servera
13         self.port = port # port servera
14         self.quota = quota*1048576 # kvota u bajtovima
15         self.run_server()
16
17
18     def run_server(self):
19         self.s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
20         self.s.setsockopt(socket.SOL_SOCKET, socket.SO_REUSEADDR, 1)
21         self.s.bind((self.adresa, self.port))
22         print("Otvaram pasivni socket na ", self.s.getsockname())
23         self.s.listen(1)
24
25         self.clients = dict()
26
27
28         # primanje novih klijenata
29         while True:
30             try:
31                 (sock, client_adresa) = self.s.accept()
32             except:
33                 print("Otpustam klijenta...")
34                 sock.close()
35                 continue
36
37
38             client_socket = ssl.wrap_socket(sock, server_side=True,\
39                 ssl_version=ssl.PROTOCOL_SSLv23,cert_reqs=ssl.CERT_NONE,\
40                 keyfile="server.key",certfile="server.crt")
41
42
43             # dokumentiranje klijenta
44             nick = client_socket.recv(20).decode()
45             if nick == "":
46                 nick = "Anonymous"
47             self.clients[client_socket] = [nick,self.quota]
48             mapa = os.getcwd()+"\\"+self.clients[client_socket][0] # stvaranje
49                 foldera za svakog klijenta
50             if not os.path.isdir(mapa):
51                 os.makedirs(mapa)
52
53
```

```

54         # stvaranje posebne niti za rad s klijentom
55         t=threading.Thread(target=self.upravljaj, args=[client_socket])
56         t.setDaemon(1)
57         t.start()
58
59
60     def upravljaj(self, client_socket):
61         print("Upravo se spojio klijent sa ", client_socket.getpeername())
62         try:
63             self.receive_files(client_socket)
64             print("Primanje podataka uspjesno zavrшено")
65         except:
66             client_socket.close()
67
68
69     def receive_files(self, client_socket):
70         chunk = client_socket.recv(84)
71         paket = pickle.loads(chunk)
72         velicina_datoteke = int(paket[1],2)
73         velicina_imena = int(paket[0],2)
74         ime = client_socket.recv(velicina_imena).decode()
75         if ime.count("\\") != 0:
76             Lista = [ime.rsplitle("\\",j)[0] for j in range(1,ime.count("\\")+1)]
77             for i in range(1,ime.count("\\")+1):
78                 mapa=os.getcwd()+"\\"+\\
79                 self.clients[client_socket][0]+"\\"+Lista[-i]
80                 if not os.path.isdir(mapa):
81                     os.makedirs(mapa)
82                 ime_datoteke = mapa+"\\"+ime.rsplitle("\\",1)[1]
83             else:
84                 ime_datoteke = self.clients[client_socket][0]+"\\"+ime
85             if self.clients[client_socket][1] - velicina_datoteke <= 0:
86                 client_socket.send("Dozvoljena memorija je puna.")
87                 return
88             self.clients[client_socket][1] = \
89                 self.clients[client_socket][1] - velicina_datoteke # smanjui kvotu
90             chunk = client_socket.recv(1024)
91             velicina_datoteke = velicina_datoteke - len(chunk)
92             if os.path.exists(ime_datoteke):
93                 novo_ime = self.save_file(ime_datoteke,1)
94                 if not novo_ime:
95                     client_socket.send("Premasili ste kvotu od 10 spremljenih stanja."
96                                     .encode())
97                     return
98                 datoteka = open(novo_ime, 'wb')
99             else:
100                 datoteka = open(ime_datoteke, 'wb')
101             while(chunk):
102                 if self.clients[client_socket][1]-velicina_datoteke<=0:
103                     client_socket.send("Dozvoljena memorija je puna.")
104                     break
105                 self.clients[client_socket][1] = \
106                     self.clients[client_socket][1] - velicina_datoteke # smanjui kvotu
107                 print("%s salje %s na server..." \
108                     %(self.clients[client_socket][0],os.path.basename(ime)))
109                 if velicina_datoteke == 0:
110                     datoteka.write(chunk)
111                 else:

```

```

111         while True:
112             datoteka.write(chunk)
113             chunk = client_socket.recv(1024)
114             velicina_datoteke = velicina_datoteke - len(chunk)
115             #print(velicina_datoteke)
116             if velicina_datoteke == 0:
117                 datoteka.write(chunk)
118                 break
119         datoteka.close()
120         chunk = client_socket.recv(84)
121         paket = pickle.loads(chunk)
122         velicina_datoteke = int(paket[1],2)
123         velicina_imena = int(paket[0],2)
124         ime = client_socket.recv(velicina_imena).decode()
125         if ime.count("\\") != 0:
126             Lista=[ime.rsplit("\\",j)[0] for j in range(1,ime.count("\\")+1)]
127             for i in range(1,ime.count("\\")+1):
128                 mapa=os.getcwd()+"\\"+ \
129                     self.clients[client_socket][0]+"\\"+Lista[-i]
130                 if not os.path.isdir(mapa):
131                     os.makedirs(mapa)
132                 ime_datoteke = mapa+"\\"+ime.rsplit("\\",1)[1]
133         else:
134             ime_datoteke=self.clients[client_socket][0]+"\\"+ime
135         if self.clients[client_socket][1]-velicina_datoteke<=0:
136             client_socket.send("Dozvoljena memorija je puna.")
137             break
138             self.clients[client_socket][1] = \
139                 self.clients[client_socket][1] - velicina_datoteke #
140                                     smanjui kvotu
141         chunk = client_socket.recv(1024)
142         velicina_datoteke = velicina_datoteke - len(chunk)
143         if os.path.exists(ime_datoteke):
144             novo_ime = self.save_file(ime_datoteke,1)
145             if not novo_ime:
146                 client_socket.send("Premasili ste kvotu od 10 spremljenih
147                                     stanja.".encode())
148             return
149             datoteka = open(novo_ime, 'wb')
150         else:
151             datoteka = open(ime_datoteke, 'wb')
152         datoteka.close()
153
154         # metoda za provjeru postoji li vec datoteka
155         def save_file(self,filename,broj):
156             if broj > 10:
157                 return ""
158             novo = "%s(%d).%s" %(filename.rsplit(".",1)[0],broj,\
159                                 filename.rsplit(".",1)[1])
160             if os.path.exists(novo):
161                 return self.save_file(filename,broj+1)
162             else:
163                 return novo
164
165
166

```

```

167
168 class BackupClient:
169     def __init__(self, remote_host, remote_port, backup_dir, time):
170         self.remote_host = remote_host
171         self.remote_port = remote_port
172         self.backup_dir = backup_dir
173         self.time = time
174         self.run_client()
175
176
177     def run_client(self):
178         sock = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
179         print("Spajam se na server na %s, port: %d" \
180               %(self.remote_host, self.remote_port))
181         try:
182             sock.connect((self.remote_host, self.remote_port))
183         except:
184             print("Spajanje sa serverom nije uspjelo...")
185             sys.exit(1)
186
187
188         self.s = ssl.wrap_socket(sock, ssl_version=ssl.PROTOCOL_SSLv3)
189
190
191         nick = input("Unesite ime: ")
192         self.s.sendall(nick.encode())
193
194
195         # nit za primanje podataka od servera
196         t_recv = threading.Thread(target = self.recv_mess)
197         t_recv.setDaemon(1)
198         t_recv.start()
199
200
201         # klijent salje datoteke na server
202         for root, directories, filenames in os.walk(self.backup_dir):
203             for file in filenames:
204                 if root != self.backup_dir:
205                     self.send_file(os.path.join(root.replace(self.backup_dir+\'
206                                                         "\\", "")), file))
207                 else:
208                     self.send_file(file)
209                     time.sleep(1)
210         print("Slanje uspjesno zavrшено...")
211
212
213         # nit koja ce pratiti promjenu datoteka
214         t_stamp = threading.Thread(target = self.timestamp)
215         t_stamp.setDaemon(1)
216         t_stamp.start()
217
218
219         status = input(">>> ")
220         if status == "exit":
221             self.s.close()
222             print("Zavrsavam sa radom...")
223
224

```

```

225
226 def timestamp(self):
227     Vrijeme = dict()
228     for root,directories,filenames in os.walk(self.backup_dir):
229         for file in filenames:
230             if root != self.backup_dir:
231                 Vrijeme[os.path.join(root.replace(self.backup_dir+"\\", ""), \
232                                     file)] = os.path.getmtime(os.path.join(root, file))
233             else:
234                 Vrijeme[file]=os.path.getmtime(self.backup_dir+"\\ "+file)
235     while True:
236         time.sleep(self.time)
237         for root, directories, filenames in os.walk(self.backup_dir):
238             for file in filenames:
239                 try:
240                     if root != self.backup_dir:
241                         if os.path.join(root.replace(self.backup_dir+"\\", ""),
242                                             file) not in Vrijeme.keys() and \
243                             os.path.join(root.replace(self.backup_dir+
244                                                         "\\ ", " "), file)[0:2] != "~$":
245                             Vrijeme[os.path.join(root.replace(self.backup_dir+
246                                                         "\\ ", " "), file)]=os.path.getmtime(os.path.\
247                                                         join(root, file))
248                             self.send_file(os.path.join(root.replace(self.\
249                                                         backup_dir+"\\ ", " "), file))
250                             print("Slanje uspjesno zavrшено...")
251                         else:
252                             if Vrijeme[os.path.join(root.replace(self.\
253                                                         backup_dir+"\\ ", " "), file)]!=os.path.\
254                                 getmtime(os.path.join(root, file)):
255                                 Vrijeme[os.path.join(root.replace(self.\
256                                                         backup_dir+"\\ ", " "), file)]=os.path.\
257                                     getmtime(os.path.join(root, file))
258                                 self.send_file(os.path.join(root.\
259                                                         replace(self.backup_dir+"\\ ", " "), file))
260                                 print("Slanje uspjesno zavrшено...")
261                     else:
262                         if file not in Vrijeme.keys() and file[0:2] != "~$":
263                             Vrijeme[file]=os.path.getmtime(self.backup_dir+
264                                                         "\\ "+file)
265                             self.send_file(file)
266                             print("Slanje uspjesno zavrшено...")
267                         else:
268                             if Vrijeme[file]!=os.path.getmtime(self.\
269                                                         backup_dir+"\\ "+file):
270                                 Vrijeme[file]=os.path.getmtime(self.\
271                                                         backup_dir+"\\ "+file)
272                                 self.send_file(file)
273                                 print("Slanje uspjesno zavrшено...")
274                 except:
275                     pass
276
277 def send_file(self, datoteka):
278     temp = open(self.backup_dir+"\\ "+datoteka, 'rb')
279     velicina_datoteke=os.path.getsize(self.backup_dir+"\\ "+datoteka)
280     prvi_paket=pickle.dumps((dec_to_bin(len(datoteka.encode()))), \

```



```

281         dec_to_bin(velicina_datoteke)))
282     self.s.sendall(prvi_paket)
283     self.s.sendall(datoteka.encode())
284     print("Saljem na server %s" %os.path.basename(datoteka))
285     chunk = temp.read(1024)
286     while(chunk):
287         self.s.sendall(chunk)
288         chunk = temp.read(1024)
289     temp.close() # zatvori kad završis sa slanjem
290
291
292     def recv_mess(self):
293         while True:
294             poruka = self.s.recv(1024)
295             if not poruka:
296                 break
297             print(poruka.decode())
298
299
300
301     def dec_to_bin(broj):
302         return bin(broj)[2:].zfill(32)
303
304
305
306
307     def main():
308         #napomena: host i port se moraju učitati iz backup_server.conf
309
310         Server=[line.strip() for line in open("backup_server.txt", 'r')]
311         host = Server[1].split(":")[1]
312         port = int(Server[2].split(":")[1])
313         kapacitet = int(Server[0].split(":")[1])
314
315         Client=[line.strip() for line in open("backup_client.txt", 'r')]
316         server_host = Client[2].split(":")[1]
317         server_port = int(Client[3].split(":")[1])
318         direktorij = Client[0].split(":",1)[1]
319         interval = int(Client[1].split(":")[1])
320
321         #poziv poslužitelja
322         if sys.argv[1] == 'server':
323             backup_server = BackupServer(host,port,kapacitet)
324
325         #poziv klijenta
326         elif sys.argv[1] == 'client':
327             # napomena: host i port se moraju učitati iz backup_client.conf
328             backup_client = BackupClient(server_host,server_port,direktorij,interval)
329
330
331     if __name__ == '__main__':
332         main()

```

Literatura

- [1] Brandon Rhodes and John Goerzen, Foundations of Python Networking Programing, 2nd Edition, Apress, 2010.
- [2] James F. Kurose and Keith W. Ross, Computer Networking: A Top-Down Aproach, 6th Edition, Pearson, 2012.