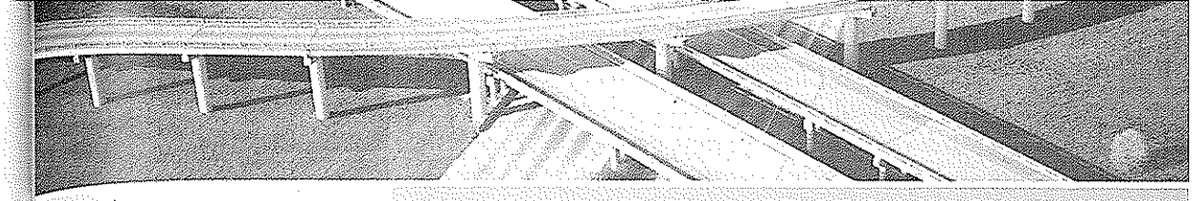
Ko vas je u profesionalnom smislu najviše impresionirao?

Bez daljeg, Klod Šenon sa MIT univerziteta – brilijantan istraživač koji je posedovao dar da svoje matematičke ideje poveže sa fizičkim svetom na izuzetno intuitivne načine. On je bio član komisije pred kojom sam branio doktorsku tezu.

Da li imate savet za studente koji tek zakoračuju u polje umrežavanja i interneta?

Internet i njegove mogućnosti su bezgranične, pune neverovatnih izazova. U njemu i te kako ima prostora za inovacije. Ne dozvolite da vas ograniči današnja tehnologija. Zakoračite u njega, zamislite šta bi moglo da bude i zatim učinite da se to i ostvari.



Aplikativni sloj

Mrežne aplikacije su *raison d'être* (razlog postojanja) računarskih mreža – bez korisnih aplikacija ne bi ni bilo razloga za projektovanje mrežnih protokola koji ih podržavaju. I zaista, od kako se internet pojavio kreirane su brojne korisne i zanimljive aplikacije. Ove aplikacije su bile pokretačka sila koja je stajala iza uspeha interneta, motivišući ljude u domaćinstvima, školama, vladama i kompanijama da internet učine sastavnim delom svojih svakodnevnih aktivnosti.

U internet aplikacije spadaju klasične tekstualne aplikacije: za elektronsku poštu, daljinski pristup računarima, prenos datoteka i forumi, koje su bile popularne sedamdesetih i osamdesetih godina. Zatim, tu je i najpopularnija aplikacija iz sredine devedesetih: *World Wide Web*, koja je u sebi sjedinjavala surfovanje vebom, pretraživanje podataka sa interneta i elektronsku prodaju. U internet aplikacije takođe spadaju dve izuzetne aplikacije koje su se pojavile krajem prošlog milenijuma – za instant poruke i za P2P razmenu datoteka. Od dvehiljadite godine videli smo eksploziju popularnih audio i video aplikacija, uključujući: govor preko interneta (VoIP) i video konferencije preko interneta kao što je Skype; distribuciju korisničkih video zapisa, kao što je YouTube; i filmovi na zahtev poput Netflix-a. Tokom ovog istog perioda videli smo takođe pojavu visoko angažovanih onlajn igrica za više igrača, uključujući *Second Life* i *World of Warcraft*. I nedavno, pojavile su se nove generacije aplikacija za društvene mreže, kao što su *Facebook* i *Twitter*, koje su kreirale zanimljive ljudske mreže na vrhu internet mreže rutera i komunikacionih linkova. Jasno, nije bilo usporavanja novih i uzbudljivih internet aplikacija. Možda će neki čitaoci ovog teksta kreirati novu generaciju sjajnih (eng. killer) internet aplikacija!

U ovom poglavlju proučavamo mrežne aplikacije sa teoretske i praktične strane. Počinjemo definisanjem ključnih pojmova aplikativnog sloja, kao što su mrežne usluge koje su neophodne aplikacijama, klijente i servere, procese i interfejse transportnog sloja. Podrobno istražujemo nekoliko mrežnih aplikacija, kao što su veb, elektronska pošta, DNS, P2P (eng. peer-to-peer) razmena datoteka (poglavlje 8 se bavi multimedijalnim aplikacijama, uključujući video tokove i govor preko interneta (VoIP). Potom prikazujemo postupak razvoja mrežnih aplikacija, koje koriste protokol TCP ili UDP. Posebno proučavamo API soket i razmatramo neke jednostavne klijentsko-serverske aplikacije koje su urađene u programskom jeziku *Python*. Osim toga, na kraju poglavlja nalazi se i nekoliko zabavnih i zanimljivih zadataka u vezi sa programiranjem soketa.

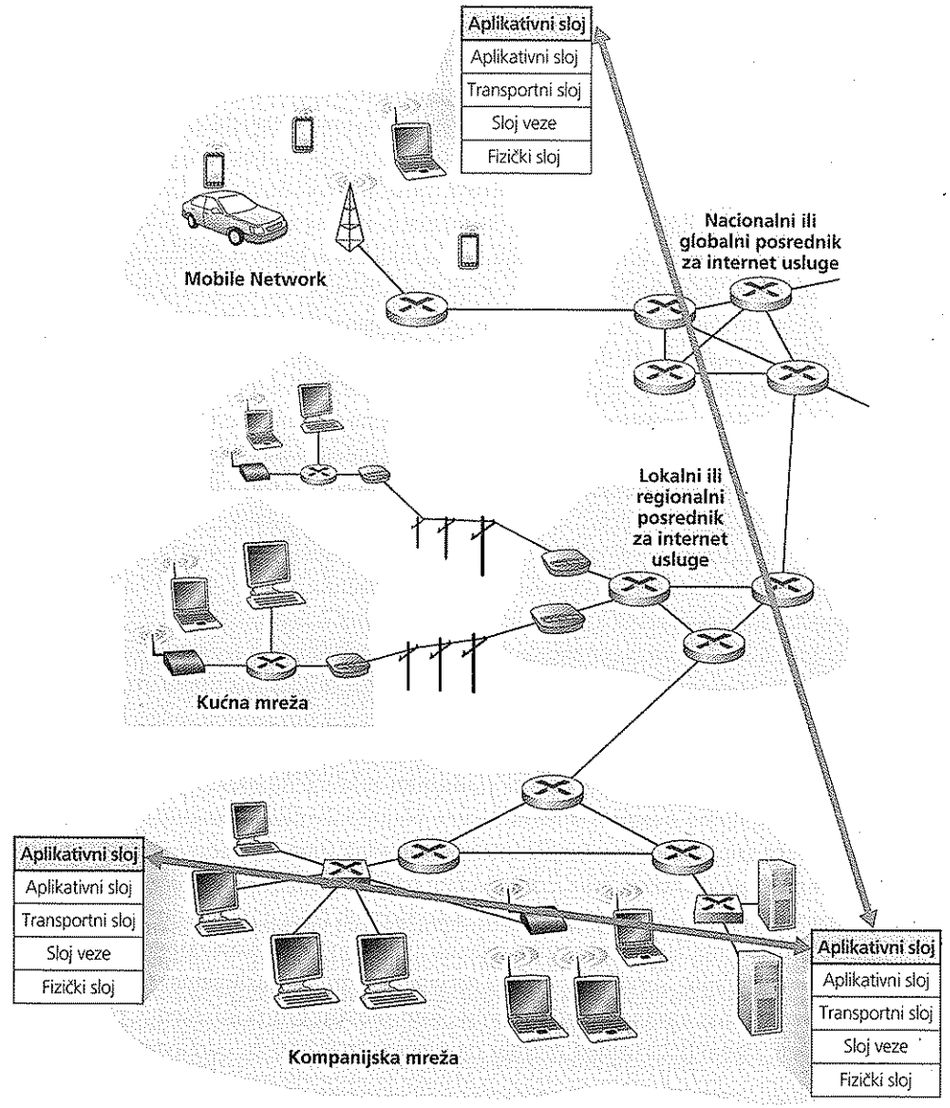
Aplikativni sloj predstavlja odlično mesto za početak našeg proučavanja protokola, pre svega zato što je u pitanju poznati teren. Sigurno su vam već dobro poznate mnoge aplikacije koje se oslanjaju na protokole o kojima ćemo govoriti. Preko njih ćemo steći predstavu o tome šta su protokoli i upoznaćemo većinu nepoznanica koje će se pojaviti i kasnije kada budemo proučavali protokole transportnog, mrežnog, kao i sloja veze.

2.1 Principi mrežnih aplikacija

Pretpostavimo da imate ideju za novu mrežnu aplikaciju. Možda će ta aplikacija predstavljati dobrobit za čovečanstvo, možda ćete njome zadiviti svog profesora, možda će vam doneti velike pare, a moguće je i da ćete jednostavno uživati u njenom razvoju. Šta god da je razlog, pogledajmo kako biste mogli da tu svoju ideju pretvorite u stvarnu mrežnu aplikaciju.

Najvažniji deo razvoja mrežne aplikacije predstavlja pisanje programa koji će se izvršavati na različitim krajnjim sistemima i međusobno komunicirati kroz mrežu. Na primer, veb aplikacija ima dva nezavisna udaljena programa koji komuniciraju jedan sa drugim: program pretraživača veba (eng. browser) koji se izvršava na korisnikovom računaru (stonom računaru, laptopu, tabletu, mobilnom telefonu i tako dalje) i program za veb server koji se izvršava na računaru koji ima ulogu veb servera. Kao drugi primer mogao bi da nam posluži P2P sistem za razmenu datoteka po kome na svakom računaru koji učestvuje u ovom sistemu postoji odgovarajući program. U ovom slučaju sasvim je moguće da su ti programi slični ili potpuno isti.

Dakle, programirajući svoju novu aplikaciju moraćete da napišete softver koji će se izvršavati na više različitih krajnjih sistema. Ovaj softver možete da napišete, na primer, korišćenjem programskog jezika C, Java ili Python. Veoma je važno to što ne morate da pišete programe koji se izvršavaju na uređajima unutar jezgra mreže, kao što su ruteri ili komutatori sloja veze. Čak i kada biste želeli da napišete aplikativni softver za ove uređaje unutar jezgra mreže, to jednostavno ne bi bilo moguće. Kao što smo naučili u poglavlju 1, što je i prikazano na slici 1.24, ključni uređaji unutar jezgra mreže ne funkcionišu na aplikativnom sloju, već samo na nižim slojevima – na mrežnom i onima koji se nalaze ispod njega. Ova osnovna postavka – tačnije, ograničavanje aplikativnog softvera samo na krajnje sisteme – kao što je prikazano na slici 2.1, zaslužna je za izuzetno brz razvoj i primenu široke lepeze mrežnih aplikacija.



Slika 2.1 ◆ Komunikacija mrežnih aplikacija obavlja se između krajnjih sistema na aplikativnom sloju

2.1.1 Arhitektura mrežnih aplikacija

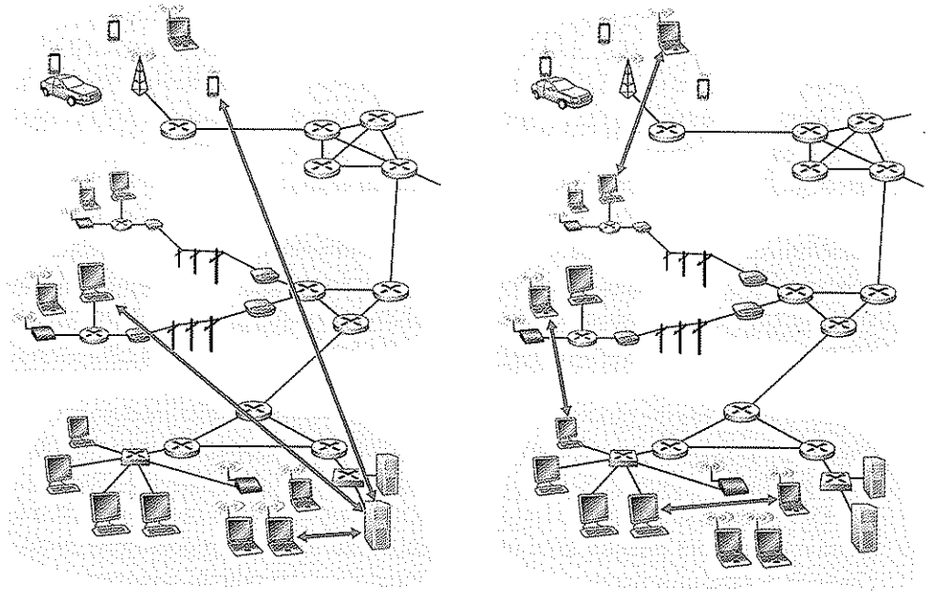
Pre nego što zaronite u pisanje programa, potrebno je da odlučite kakva će biti arhitektura vaše aplikacije. Uvek imajte u vidu to da se arhitektura aplikacije bitno razlikuje od arhitekture mreže (npr. podsećamo vas na petoslojnu arhitekturu interneta o kojoj smo govorili u poglavlju 1). Sa stanovišta programera aplikacije, arhitektura mreže je nepromenljiva i aplikacijama nudi određeni skup usluga. **Arhitekturu aplikacije**, s druge strane, projektuje njen programer i ona određuje način na koji se aplikacija organizuje na različitim krajnjim sistemima. Prilikom izbora arhitekture aplikacije, programer se opredeljuje za jedno od dva preovlađujuća rešenja koja se koriste za arhitekturu savremenih mrežnih aplikacija: klijentsko-serverska arhitektura ili P2P arhitektura.

U **klijentsko-serverskoj arhitekturi**, postoji uvek dostupan računar, nazvan *server*, čije usluge traži mnogo drugih računara, koji se nazivaju *klijenti*. Klasičan primer ovakve arhitekture predstavlja veb aplikacija u kojoj uvek dostupan server opslužuju zahteve pretraživača veba koji se izvršavaju na računarima klijentima. Kada od računara klijenta primi zahtev za određeni objekat, veb server odgovara slanjem traženog objekta do računara klijenta. Skrećemo vam pažnju na to da u klijentsko-serverskoj arhitekturi, klijenti ne komuniciraju neposredno jedan sa drugim; recimo, u veb aplikaciji ne postoji neposredna komunikacija dva pretraživača veba. Drugu važnu karakteristiku ove arhitekture predstavlja činjenica da server ima stalnu, dobro poznatu mrežnu adresu, koju zovemo IP adresa, (uskoro ćemo više govoriti o tome). Pošto server ima stalnu, dobro poznatu adresu i uvek je dostupan, klijent uvek može da ga kontaktira, šaljući odgovarajući paket na IP adresu servera. U poznatije klijentsko-serverske aplikacije spadaju: veb, FTP, Telnet i e-pošta. Klijentsko-serverska arhitektura prikazana je na slici 2.2(a).

U klijentsko-serverskim aplikacijama se često dešava da jedan server ne može da se izbori sa svim zahtevima svojih klijenata. Na primer, na popularnim veb lokacijama za društveno umrežavanje korisnika mogući su zastoji ukoliko bi se samo jedan server koristio za obradu svih zahteva klijenata. Upravo zato se, u klijentsko-serverskoj arhitekturi, koristi **centar podataka** u kome se nalazi veliki broj računara da bi se dobio moćan virtuelni server. Najpopularniji internet servisi – poput pretraživača (npr. *Google* i *Bing*), internet trgovine (npr. *Amazon* i *e-Bay*), veb e-pošte (npr. *Gmail* i *Yahoo Mail*), društvenog umrežavanja (npr. *Facebook* i *Twitter*) – koriste jedan ili više centara podataka. Kao što smo rekli u poglavlju 1.3.3 *Google* ima 30 do 50 centara podataka raspoređenih svuda po svetu, koji zajedno upravljaju pretragom, *YouTube* servisima, *Gmail* servisima i drugim uslugama. Centar podataka može da ima stotine hiljada servera, koji moraju biti priključeni i održavani. Pored toga, dobavljači usluga moraju da plaćaju troškove za održavanje linkova i propusni opseg koje koriste za slanje podataka iz njihovih centara podataka.

U **P2P arhitekturi**, postoji neznatna (ili nikakva) zavisnost od namenskih servera u centrima podataka. Umesto toga, aplikacije koriste direktnu komunikaciju između parova *ravnopravnih računara* (*engl.* peers). Ovi računari nisu u posedu dobavljača usluga, već su to stoni ili laptop računari kojima upravljaju korisnici, pri čemu se većina ovih računara nalazi u domaćinstvima, na univerzitetima i u

kancelarijama. Budući da ravnopravni računari komuniciraju bez posredovanja posebnog servera, ova arhitektura se naziva arhitekturom ravnopravnih korisnika (eng. peer-to-peer, P2P). Većina najpopularnijih savremenih aplikacija sa najgušćim saobraćajem zasniva se na P2P arhitekturi. U ove aplikacije spadaju one za razmenu datoteka (npr. *Bit Torrent*), ubrzano preuzimanje zasnovano na mreži ravnopravnih računara (npr. *Xunlei*), internet telefonija (npr. *Skype*) i internet televizija (npr. *Kankan* i *PPstream*). P2P arhitektura prikazana je na slici 2.2(b). Podsećamo da neke aplikacije imaju hibridnu arhitekturu, objedinjavajući klijentsko-serverske i P2P elemente. Na primer, mnoge aplikacije za razmenu instant poruka koriste servere za čuvanje podataka o internet adresama korisnika, dok se poruke korisnika razmenjuju direktno između računara (i ne prolaze kroz posredne servere).



(a) Klijentsko-serverska arhitektura (b) P2P arhitektura

Slika 2.2 ◆ (a) Klijentsko-serverska arhitektura (b) P2P arhitektura

Jedna od najboljih osobina P2P arhitekture jeste njena **prilagodljivost**. Na primer, u aplikaciji za P2P razmenu datoteka, računari koji traže datoteke stvaraju dodatno opterećenje mreže, ali istovremeno proširuju kapacitete sistema za distribuciju datoteka drugim računarima. P2P arhitektura ne zahteva dodatna ulaganja, pošto za nju obično nije potrebna značajnija serverska infrastruktura i veća propusna moć (za razliku od klijentsko-serverskog dizajna sa centrima podataka). Međutim, budućnost P2P aplikacija suočava se sa tri velika izazova:

1. *Prijateljski nastrojen prema posredniku za internet usluge.* Većina lokalnih posrednika za internet usluge (uključujući posrednike za digitalnu pretplatničku liniju DSL i kablovske posrednike) su projektovale svoje mreže za

„asimetrično" korišćenje propusnog opsega, to jest više protoka za nizvodni, a manje za uzvodni saobraćaj. Međutim, P2P video tokovi i aplikacije za distribuciju datoteka pomeraju uzvodni sabraćaj od servera do lokalnog posrednika za internet usluge, i time značajno opterećuju tog posrednika. Buduće P2P aplikacije bi trebalo da budu dizajnirane tako da budu prijateljski nastrojene prema posrednicima za internet usluge [Xie 2008].

2. *Bezbednost.* S obzirom da su P2P aplikacije u velikoj meri distribuirane i da su otvorene prirode, njihova bezbednost može biti izazov [Doucer 2002; Yu 2006; Liang 2006; Naoumov 2006; Dhungel 2008; LeBlond 2011].
3. *Podsticaji.* Uspeh budućih P2P aplikacija zavisi i od toga da li će se korisnici ubediti da aplikacijama dobrovoljno ponude propusni opseg, skladište i računske resurse, što predstavlja izazov podsticajnog dizajna [Feldman 2005; Piatek 2008; Aperjis 2008; Liu 2010].

2.1.2 Komunikacija između procesa

Pre nego što počnete da razvijate svoju mrežnu aplikaciju potrebno je da, bar u najosnovnijim crtama, znate na koji način komuniciraju programi koji se nalaze na različitim krajnjim sistemima. U žargonu operativnih sistema ti programi se nazivaju **procesi**. Proces možete da zamislite i kao program koji se izvršava na krajnjem sistemu. Ukoliko se procesi izvršavaju na istom krajnjem sistemu, oni međusobno komuniciraju međuprocesnom komunikacijom, koristeći pravila kojima upravlja operativni sistem krajnjeg sistema. Međutim, u ovoj knjizi nećemo se baviti komuniciranjem između procesa na istom računaru, već time kako međusobno komuniciraju procesi koji se izvršavaju na *različitim* računarima (čak i sa različitim operativnim sistemima).

Procesi koji se izvršavaju na dva različita krajnja sistema međusobno komuniciraju razmenom **poruka** kroz računarsku mrežu. Predajni proces pravi poruke i šalje ih u mrežu; prijemni proces prima te poruke i po potrebi odgovara šaljući poruke nazad. Na slici 2.1 prikazani su procesi koji međusobno komuniciraju na aplikativnom sloju petoslojnog skupa protokola.

Klijentski i serverski procesi

Mrežne aplikacije sastoje se od parova procesa koji jedni drugima šalju poruke kroz mrežu. Na primer, u slučaju veb aplikacije, proces veb pretraživača klijenta razmenjuje poruke sa procesom veb servera. U sistemu P2P razmeni datoteka, datoteke se prenose od procesa na jednom ravnopravnom računaru do procesa na drugom ravnopravnom računaru. Za svaki par procesa koji međusobno komunicira obično jedan od ova dva procesa nazivamo **klijent**, a drugi **server**. Za veb aplikaciju, veb pretraživač je klijentski proces, a veb server je serverski proces. U P2P razmeni datoteka, računar koji preuzima datoteku naziva se klijent, a računar koji predaje datoteku naziva se server.

Verovatno ste primetili da u nekim aplikacijama, kao što je P2P razmena datoteka, proces može da istovremeno bude i klijent i server. Zaista, u sistemu za P2P razmenu datoteka isti proces može i da preuzima i da predaje datoteke. Ipak, u okviru komunikacione sesije između dva procesa jedan od njih uvek možemo da nazovemo klijent, a drugi server. Klijentske i serverske procese definišemo na sledeći način:

U okviru komunikacione sesije između dva procesa, proces koji pokreće komunikaciju (to jest, prvi kontaktira drugi proces na početku sesije) označava se kao klijent. Proces koji čeka na poziv da bi uspostavio komunikaciju naziva se server.

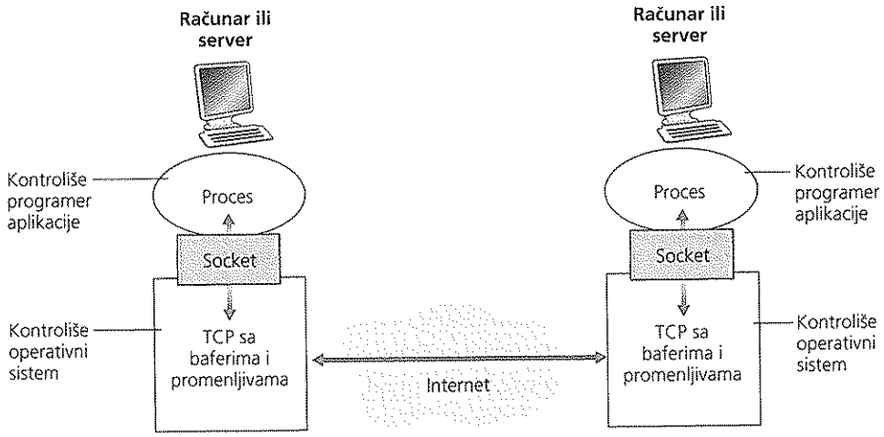
U slučaju veba, proces veb pretraživača pokreće komunikaciju sa procesom veb servera; znači, proces veb pretraživača je klijent, a proces veb servera je server. Kod P2P razmene datoteka, kada računar A zatraži od računara B da mu pošalje određenu datoteku, računar A je klijent, a računar B je server u okviru ove komunikacione sesije. Kada nema sumnje u pogledu ovakve podele uloga, ponekad ćemo koristiti i izraze „klijentska strana" i „serverska strana aplikacije". Na kraju poglavlja proći ćemo kroz jednostavan kôd klijentske i serverske strane mrežne aplikacije.

Interfejs između procesa i računarske mreže

Kao što smo već rekli, većinu aplikacija sačinjavaju parovi procesa koji međusobno komuniciraju, pri čemu dva procesa u svakom paru šalju poruke jedan drugom. Sve poruke poslate od jednog procesa do drugog moraju da prođu kroz mrežu između njih. Procesi šalju poruke u mrežu i primaju poruke iz mreže preko softverskog interfejsa koji se naziva **soket** (engl. *socket*). Uzmimo za primer poređenje koje će nam pomoći da bolje razumemo procese i sokete. Proces odgovara kući, a njegov soket odgovara vratima na toj kući. Kada neki proces želi da pošalje poruku drugom procesu na drugom računaru, on tu poruku izbacuje u mrežu kroz svoja vrata (soket). Pri tome proces koji šalje pretpostavlja da s druge strane njegovih vrata postoji transportna infrastruktura koja će poruku preneti do vrata odredišnog procesa. Pošto stigne do odredišnog računara, poruka prolazi kroz vrata (soket) procesa primaoca, koji zatim postupa u skladu sa tom porukom.

Na slici 2.3 prikazana je komunikacija soketa između dva procesa koji komuniciraju preko interneta. (Na ovoj slici pretpostavlja se da je transportni protokol koji procesi koriste internetov TCP protokol.) Kao što možete da vidite, soket je posrednik između aplikativnog i transportnog sloja na računaru. Naziva se još i **programski interfejs aplikacije** (Application Programming Interface, **API**) između određene aplikacije i mreže, zato što je soket programski interfejs pomoću koga se prave mrežne aplikacije. Programer aplikacije može da utiče na sve ono što se događa na strani aplikativnog sloja soketa, ali veoma malo na ono što se događa na strani transportnog sloja soketa. Jedino na šta programer može da utiče na strani transportnog sloja je: (1) izbor transportnog protokola i 2) mogućnost da postavi nekoliko parametara transportnog sloja, kao što su maksimalna veličina bafera ili maksimalna veličina segmenta (o tome govorimo u poglavlju 3). Pošto programer

izabere transportni protokol (ukoliko uopšte ima mogućnost izbora), aplikaciju pravi korišćenjem usluga transportnog sloja, koje pruža taj protokol. Sokete podrobnije obrađujemo u odeljku 2.7.



Slika 2.3 ◈ Aplikativni procesi, soketi i transportni protokol na kome se zasnivaju

Adresiranje procesa

Da bismo poštansko pismo poslali na određenu destinaciju, destinacija mora da ima adresu. Slično, kao što je da bi proces koji se izvršava na jednom računaru slao pakete procesu koji se izvršava na drugom računaru neophodno da prijemni proces ima adresu. Da bi se identifikovao prijemni proces, potrebno je odrediti dva podatka: (1) adresu računara i (2) identifikator koji određuje prijemni proces na odredišnom računaru.

Na internetu se računar identifikuje svojom **IP adresom**. O IP adresama ćemo više govoriti u poglavlju 4. Za sada, sve što bi trebalo da znate je da je IP adresa 32-bitna veličina koja jedinstveno identifikuje računar. Pored toga što zna adresu računara kome je poruka namenjena, predajni proces mora takođe da indentifikuje prijemni proces (tačnije, prijemni soket) koji se izvršava na tom računaru. Ova informacija je potrebna jer, generalno, računar može da izvršava brojne mrežne aplikacije. Odredišni **broj porta** koristi se za ovu svrhu. Popularnim aplikacijama dodeljeni su određeni brojevi porta. Na primer, veb server se identifikuje brojem porta 80. Proces servera za e-poštu (koji koristi SMTP protokol) se idenfikuje brojem porta 25. Lista dobro poznatih brojeva portova za sve standardne internet protokole može da se nađe na adresi http://www.iana.org. Brojeve portova detaljnije ćemo ispitivati u poglavlju 3.

2.1.3 Transportne usluge koje su dostupne aplikacijama

Podsećamo da soket predstavlja interfejs između aplikativnog procesa i protokola transportnog sloja. Aplikacija na predajnoj strani šalje poruke kroz soket. Za prenos poruka do soketa prijemnog procesa odgovoran je protokol transportnog sloja.

U mnogim mrežama, uključujući i internet, postoji više transportnih protokola. Prilikom programiranja aplikacija morate da izaberete jedan od raspoloživih protokola transportnog sloja. Kako da se odlučite? Najčešće, tako što pažljivo proučite usluge koje nude raspoloživi protokoli transportnog sloja i onda izaberete protokol sa uslugama koje najviše odgovaraju potrebama vaše aplikacije. Ovo liči na izbor između voza i aviona za putovanje između dva grada. Možete da birate jedan od njih, a svaki od ova dva načina transporta nudi različite usluge. (Na primer, pogodnost voza je što polazite iz centra grada i stižete u centar grada, dok je pogodnost aviona mnogo kraće vreme putovanja.)

Koje usluge protokol transportnog sloja može da ponudi aplikaciji koja ga koristi? Uopšteno govoreći, moguće usluge mogu da se razvrstaju na osnovu četiri parametra: pouzdan prenos podataka, propusna moć, vremenska ograničenja i bezbednost.

Pouzdan prenos podataka

Kao što smo rekli u poglavlju 1, paketi se mogu izgubiti unutar računarske mreže. Na primer, paket ne može da stane u prepunjeni bafer rutera, ili ga računar ili ruter mogu odbaciti, pošto ustanove da je neki njegov deo oštećen. Za mnoge aplikacije kao što su: e-pošta, prenos datoteka, daljinsko pristupanje računarima, prenos veb dokumenata i finansijske aplikacije – gubici podataka mogu da imaju nesagledive posledice (u poslednjem navedenom slučaju, svejedno da li po banku ili po korisnika!). Stoga se, kao podrška tim aplikacijama, mora učiniti nešto čime se garantuje da će podaci koje šalje jedna strana aplikacije biti ispravno i potpuno isporučeni drugoj strani aplikacije. Ukoliko neki protokol nudi takvu uslugu garantovane isporuke podataka, za takav protokol se kaže da obezbeđuje **pouzdan prenos podataka**. Jedna od važnijih usluga koje protokol transportnog sloja može da ponudi aplikaciji je pouzdan prenos podataka između procesa. Kada transportni protokol obezbeđuje ovu uslugu, predajni proces samo propušta svoje podatke u soket i pouzdano zna da će ti podaci bez grešaka stići do prijemnog procesa.

U slučaju kada protokol transportnog sloja ne obezbeđuje pouzdan prenos podataka može se desiti da poslati podaci nikada ne stignu do prijemnog procesa. To može da bude prihvatljivo za **aplikacije koje su tolerantne na gubitke**, među kojima su posebno značajne multimedijalne aplikacije za prenos kao što su aplikacije za prnos audio ili video zapisa u realnom vremenu, koje do izvesne mere mogu da podnesu gubitak podataka. U ovim multimedijalnim aplikacijama posledica gubitka podataka može da bude samo mali poremećaj audio ili video zapisa – što ne mora da bude strašno.

Propusna moć

U poglavlju 1 upoznali smo pojam raspoložive propusne moći koja predstavlja, u smislu komunikacije između dva procesa na određenoj putanji kroz mrežu, brzinu kojom predajni proces može da isporučuje bitove prijemnom procesu. Pošto se propusni opseg na određenoj putanji kroz mrežu deli sa drugim sesijama koji tu putanju koriste za svoju komunikaciju, a pošto se broj tih sesija stalno menja, raspoloživa propusna moć tokom vremena se menja. Ovo zapažanje prirodno navodi na drugu uslugu koju bi protokol transportnog sloja mogao da ponudi – tačnije, garantovanu propusnu moć određene brzine prenosa. Uz takvu uslugu, aplikacija bi mogla da zahteva garantovanu propusnu moć od r bita u sekundi, a transportni protokol bi osigurao da raspoloživa propusna moć uvek bude najmanje r bita u sekundi. Takva garantovana propusna moć je važna za mnoge aplikacije. Na primer, aplikaciji za internet telefoniju koja kodira glas brzinom 32 kb/s neophodno je da se slanje podataka kroz mrežu i njihova isporuka odredišnoj aplikaciji odvijaju tom brzinom. Ukoliko transportni protokol ne može da obezbedi ovu propusnu moć, aplikacija bi morala da koduje manjom brzinom (i dobije dovoljno propusne moći da održi ovu sporiju brzinu kodovanja), ili bi moralada odustane, pošto je recimo polovina potrebne propusne moći za prijem nedovoljna ili nekorisna toj aplikaciji za internet telefoniju. Za aplikacije koje zahtevaju određenu propusnu moć kaže se da su **zavisne od propusnog opsega**. Mnoge savremene multimedijalne aplikacije zavise od propusnog opsega, mada neke od njih koriste adaptivne tehnike kodovanja kojom se koduju video i glas brzinama koje odgovaraju trenutno raspoloživoj propusnoj moći.

Dok aplikacije zavisne od propusnog opsega zahtevaju određenu propusnu moć, **elastične aplikacije** mogu da koriste onoliko propusne moći koliko im je dostupno. E-pošta, prenos datoteka i prenos veb dokumenta spadaju u elastične aplikacije. Naravno, što više propusne moći, tim bolje! Ne kaže se uzalud da se ne može biti previše bogat, previše vitak i imati previše propusne moći!

Vremenska ograničenja

Protokol transportnog sloja takođe može da ponudi garanciju za ispunjenje vremenskih ograničenja. Kao i u slučaju garancija za propusnu moć, postoji više načina i oblika kojima se garantuje ispunjenje vremenskih ograničenja. Primer ovakve garancije je recimo to da svaki bit koji pošiljalac prebaci u svoj soket za najkasnije 100 milisekundi stigne u soket primaoca. Takva usluga neophodna je interaktivnim aplikacijama u realnom vremenu, kao što su: internet telefonija, virtuelno okruženje, telekonferencije i igrice za više igrača, koje zbog toga imaju vremenska ograničenja, koja se tiču isporuke podataka, čime postaju funkcionalne i upotrebljive. (Pogledajte poglavlje 7 [Gauthier 1999; Ramjee 1994].) Duža kašnjenja kod internet telefonije, recimo, stvaraju neprirodne pauze u razgovoru; kod igara sa više igrača ili virtuelnih interaktivnih okruženja, duže kašnjenje između akcije i reakcije okruženja (recimo, od igrača na drugom kraju veze) umanjuju realističnost date aplikacije. Za aplikacije koje ne funkcionišu u realnom vremenu uvek je bolje da je kašnjenje što kraće, ali tu ne postoje nikakvi striktni zahtevi.

Bezbednost

Konačno, transportni protokol može aplikacijama da ponudi jednu ili više bezbednosnih usluga. Na primer, transportni protokol može da šifruje sve podatke koje šalje predajni proces na predajnom računaru, a protokol transportnog sloja na prijemnom računaru može da dešifruje te podatke pre nego što ih isporuči prijemnom procesu. Zahvaljujući takvoj usluzi procesi uspostavljaju poverljivu komunikaciju, čak i u slučaju da se na neki način prisluškuju podaci između predajnog i prijemnog procesa. Takođe, transportni protokol može da ponudi ostale bezbednosne usluge pored tajnosti, kao što su obezbeđivanje integriteta podataka i provera autentičnosti krajnje tačke, teme o kojima podrobnije govorimo u poglavlju 8.

2.1.4 Transportne usluge koje nudi internet

Do sada smo samo uopšteno razmatrali transportne usluge koje bi računarske mreže *mogle* da ponude. Sada ćemo biti malo određeniji i govorićemo o vrstama usluga koje internet nudi. Internet (i uopšte TCP/IP mreže) aplikacijama nude dva transportna protokola: UDP i TCP. Kada (kao programer aplikacije) pravite novu internet aplikaciju, jedna od prvih odluka koju morate da donesete je da li ćete koristiti UDP ili TCP protokol. Svaki od njih nudi drugačiji skup usluga aplikacijama koje ga koriste. Na slici 2.4 prikazane su usluge koje su neophodne nekim izabranim aplikacijama.

Aplikacija	Gubici podataka	Propusni opseg	Vremenska ograničenja
Prenošenje datoteke / preuzimanje	Bez gubitaka	Elastičan	Nevažna
E-pošta	Bez gubitaka	Elastičan	Nevažna
Veb dokumenta	Bez gubitaka	Elastičan (nekoliko kb/s)	Nevažna
Internet telefonija / video konferencije	Toleriše gubitke	Zvuk: nekoliko kb/s do 1 Mb/s Video: 10 kb/s do 5 Mb/s	Postoje: stotinak msec
Snimljeni zvučni / video zapisi	Toleriše gubitke	Isto kao gore	Postoje: nekoliko sekundi
Interaktivne igrice	Toleriše gubitke	Nekoliko kb/s do 10 kb/s	Postoje: stotinak msec
Instant poruke	Bez gubitaka	Elastičan	i da i ne

Slika 2.4 ◆ Zahtevi izabranih mrežnih aplikacija

Usluge TCP protokola

Model usluga TCP protokola obuhvata uslugu sa uspostavljanjem veze i uslugu pouzdanog prenosa podataka. Kada neka aplikacija pokrene TCP protokol kao svoj transportni protokol, ona od TCP protokola dobija obe ove usluge.

- *Usluga sa uspostavljanjem veze.* TCP protokol razmenjuje kontrolne informacije transportnog sloja između klijenta i servera *pre* nego što počne protok poruka na aplikativnom nivou. Ovim postupkom koji se zove rukovanje (eng. handshaking) upozoravaju se klijent i server u cilju pripreme za razmenu paketa. Po završetku faze rukovanja kaže se da je uspostavljena **TCP veza** između soketa dva procesa. Reč je o punoj dupleksnoj vezi u kojoj oba procesa mogu istovremeno da šalju svoje poruke. Kada završi sa slanjem poruka, aplikacija mora da raskine vezu. U poglavlju 3 podrobnije razmatramo uslugu sa uspostavljanjem veze i način na koji se ostvaruje.

BEZBEDNOST PRE SVEGA

BEZBEDNOST TCP PROTOKOLA

TCP i UDP protokoli ne nude uslugu šifrovanja podataka – podaci koje predajni proces prenosi u svoj soket nepromenjeni putuju kroz mrežu do odredišnog procesa. Tako, na primer, ukoliko predajni proces pošalje lozinku u obliku otvorenog teksta (tj. nešifrovano) u svoj soket, ova će lozinka putovati svim linkovima između pošiljaoca i primaoca, pri čemu je moguće da bude uhvaćena i otkrivena na bilo kom linku između. Pošto su privatnost podataka i ostala sigurnosna pitanja koja se tiču bezbednosti sve važnija za većinu aplikacija, internet zajednica razvila je nadogradnju TCP protokola, pod nazivom **SSL** (Secure Socket Layer – sloj bezbednih soketa). TCP protokol sa SSL nadogradnjom, pored toga što radi sve ono što radi i tradicionalni TCP protokol, pruža i najvažnije bezbednosne usluge od procesa do procesa, kao što su: šifrovanje, zaštita integriteta podataka i provera autentičnosti krajnje tačke. Naglašavamo da SSL nije treći internet transportni protokol, istog nivoa kao TCP i UDP, već samo nadogradnjaTCP protokola, pri čemu je ta nadogradnja implementirana na aplikativnom sloju. U suštini, da bi neka aplikacija koristila SSL uslugu, potrebno je da se SSL kôd (postojeće, visoko optimizovane biblioteke i klase) ubaci u klijentsku i serversku stranu aplikacije. SSL ima sopstveni API priključak koji je sličan tradicionalnom API soketuTCP protokola. Kada neka aplikacija koristi SSL, predajni proces prosleđuje podatke u obliku otvorenog tekstakroz SSL soket; SSL na predajnom računaru potom šifruje te podatke i tako šifrovane podatke propušta u TCP soket. Šifrovani podaci putuju preko interneta do TCP soketa prijemnog procesa. Prijemni soket prosleđuje šifrovane podatke do SSL koji ih dešifruje. Konačno, SSL prosleđuje podatke u obliku otvorenog teksta preko svog SSL soketa do prijemnog procesa. SSL podrobnije obrađujemo u poglavlju 8.

- *Usluga pouzdanog transporta.* Procesi koji razmenjuju podatke koristeći TCP protokol, mogu da budu sigurni da će svi podaci biti isporučeni bez grešaka i pravilnim redosledom. Kada jedna strana aplikacije prosledi niz bajtova u svoj soket, ona može da računa na to da će TCP protokol isporučiti isti taj niz bajtova prijemnom soketu, bez gubitaka ili dupliranja bajtova.

TCP protokol obuhvata i mehanizam za kontrolu zagušenja, uslugu koja je korisnija internetu nego procesima koji razmenjuju podatke. TCP mehanizam za kontrolu zagušenja usporava predajni proces (klijent ili server), ukoliko dođe do zagušenja u mreži između pošiljaoca i primaoca. Kao što ćemo videti u poglavlju 3, TCP mehanizam za kontrolu zagušenja takođe nastoji da ograniči svaku TCP vezu na njihov ravnopravni udeo u propusnom opsegu mreže.

Usluge UDP protokola

UDP je pojednostavljeni transportni protokol koji nudi samo najosnovnije usluge. Protokol UDP ostvaruje se bez uspostavljanja veze, pa nema potrebe za rukovanjem između dva procesa pre početka komunikacije. UDP pruža uslugu nepouzdanog prenosa podataka – to jest, kada neki proces preda poruku u soket protokola UDP, *nema* garancije da će ta poruka zaista stići do prijemnog procesa. Osim toga, poruke koje stignu do odredišta mogu da stignu po poremećenom redosledu.

UDP protokol nema mehanizam za kontrolu zagušenja, tako da predajna strana UDP protokola može da ubacuje podatke u sloj ispod sebe (mrežni sloj) brzinom kojom želi. (Međutim, imajte na umu da stvarna propusna moć od jednog do drugog kraja može da bude manja od ove brzine zbog ograničenog propusnog opsega linkova koji se nalaze između ili zbog zagušenja.)

Usluge koje ne nude transportni protokoli interneta

Usluge koje transportni protokoli mogu da ponude razvrstali smo na osnovu četiri parametra: pouzdan prenos podataka, propusna moć, vremenska ograničenja i bezbednost. Koje od ovih usluga nude TCP i UDP protokoli? Već smo napomenuli da TCP nudi pouzdan prenos podataka od jednog do drugog kraja. Takođe, znamo da se TCP protokol na aplikativnom nivou može lako nadograditi SSL uslugom, kako bi ponudio i bezbednosne usluge. Ali u našem kraćem opisu TCP i UDP protokola namerno nije spomenuto garantovanje propusne moći i ispunjenje vremenskih ograničenja – usluge koje savremeni transportni protokoli interneta *ne* nude. Da li to znači da se aplikacije kojima je pravovremena isporuka podataka važna, kao što je internet telefonija, ne mogu koristiti u savremenom internetu? Odgovor je naravno ne – na internetu se ovakve aplikacije koriste već godinama. Ove aplikacije obično rade prilično dobro, zato što su napravljene tako da se izbore, u meri koliko je to moguće, sa nepostojanjem ovakvih garancija. Nekoliko trikova koji se za to koriste istražićemo u poglavlju 7. Ipak, mudro zamišljene aplikacije imaju svoja ograničenja kada je kašnjenje značajno,ili je propusna moć od jednog do drugog kraja ograničena.Ukratko, savremeni internet može često da ponudi zadovoljavajuće usluge aplikacijama osetljivim na vremenska ograničenja, ali ne nudi nikakve garancije što se tiče vremenskih ograničenja ili propusnog opsega.

Aplikacija	Protokol aplikativnog sloja	Transportni protokol koji se koristi
E-pošta	SMTP [RFC 5321]	TCP
Daljinski pristup	Telnet [RFC 854]	TCP
Veb	HTTP [RFC 2616]	TCP
Prenos datoteka	FTP [RFC 959]	TCP
Protok multimedijalnih podataka u realnom vremenu	HTTP (npr. YouTube)	TCP
Internet telefonija	SIP [RFC 3261], RTP [RFC 3550] ili vlasnički protokoli (npr. Skype)	UDP ili TCP

Slika 2.5 ◆ Popularne internet aplikacije, njihovi protokoli aplikativnog sloja i transportni protokoli na kojima se zasnivaju

Na slici 2.5 navedeni su transportni protokoli koje koriste neke popularne internet aplikacije. Vidimo da aplikacije za elektronsku poštu, daljinsko pristupanje, veb i prenos datoteka koriste TCP. Za ove aplikacije izabran je TCP protokol pre svega zato što obezbeđuje uslugu pouzdanog prenosa podataka, kojom se garantuje da će svi podaci na kraju stići do svog odredišta. S obzirom da aplikacije internet telefonije (kao što je *Skype*) često tolerišu određeni gubitak podataka, ali zahtevaju minimalne brzine da bi bile efikasne, programeri aplikacija internet telefonije radije izvršavaju svoje aplikacije pomoću UDP protokola i na taj način zaobilaze mehanizam za kontrolu zagušenja i veliko zaglavlje paketa TCP protokola. Međutim, pošto su mnoge mrežne barijere (većina tipova)konfigurisane da blokiraju UDP saobraćaj, aplikacije internet telefonije su često dizajnirane tako da koriste TCP protokol kao rezervnu varijantu, u slučaju da UDP komunikacija zakaže.

2.1.5 Protokoli aplikativnog sloja

Upravo smo naučili da mrežni procesi međusobno komuniciraju slanjem poruka kroz sokete. Ali, kako su te poruke strukturirane? Šta znače pojedina polja u tim porukama? Kada procesi šalju poruke? Ova pitanja dovode nas u svet protokola aplikativnog sloja. **Protokol aplikativnog sloja** definiše kako procesi neke aplikacije, koji se izvršavaju na različitim krajnjim sistemima, razmenjuju poruke. Tačnije, protokol aplikativnog sloja definiše:

- tipove poruka koje se razmenjuju, na primer, poruke zahteva i poruke odgovora;
- sintaksu za različite vrste poruka, kao što su polja u poruci i kako se ta polja razlikuju;
- značenje tih polja, to jest, značenje informacija u poljima;
- pravila po kojima se određuje kada i kako neki proces šalje poruke i odgovara na njih.

Pojedini protokoli aplikativnog sloja određeni su RFC dokumentima i stoga su javno dostupni. Na primer, protokol aplikativnog sloja koji se koristi za veb HTTP (HyperText Transfer Protocol [RFC 2616] – protokol za prenos hiperteksta) dostupan je u obliku RFC dokumenta. Ukoliko programer veb pretraživača poštuje pravila RFC dokumenta za HTTP protokol, pretraživač će moći da preuzima veb stranice sa svih veb servera koji poštuju pravila RFC dokumenta za HTTP protokol. Mnogi drugi protokoli aplikativnog sloja su vlasnički i namerno nisu javno dostupni. Na primer, *Skype* koristi vlasničke protokole aplikativnog sloja.

Veoma je važno razlikovati mrežne aplikacije i protokole aplikativnog sloja. Protokoli aplikativnog sloja samo su jedan od delova mrežne aplikacije (premda, veoma važan deo aplikacije s naše tačke gledišta!). Pogledajmo nekoliko primera. Veb je klijentsko-serverska aplikacija koja korisnicima omogućava da na zahtev dobiju dokumente sa veb servera. Veb aplikacija se sastoji od mnogo komponenti, uključujući standard za format dokumenta (to jest, HTML), veb pretraživača (na primer, *Firefox* i *Microsoft Internet Explorer*), veb servere (na primeri, *Apache* ili *Microsoft* serveri) i protokol aplikativnog sloja. Veb protokol aplikativnog sloja, HTTP, definiše format i redosled poruka koje se razmenjuju između pretraživača i veb servera. Stoga je HTTP samo jedan deo (nesumnjivo, vrlo važan deo) veb aplikacije. Drugi primer je internet aplikacija za e-poštu koja takođe ima mnogo komponenti, uključujući server za e-poštu na kome se nalazi poštansko sanduče za svakog korisnika (mailbox); klijentske programe za e-poštu (kao što je *Microsoft Outlook*) koji omogućuju korisnicama da čitaju i pišu e-poštu; standrad za definisanje strukrutre poruke e-pošte; i protokol aplikativnog sloja koji definiše kako se poruke prenose između servera, kako se poruke prenose između servera i klijentskihprograma za elektronsku poštu i kako se sadržaj zaglavlja poruke tumači. Osnovni protokol aplikativnog sloja za razmenu elektronske pošte je SMTP (Simple Mail Transfer Protocol – jednostavan protokol za prenos elektronske pošte) [RFC 5321]. Stoga, osnovni protokol aplikativnog sloja za razmenu e-pošte, SMTP, samo je jedan (mada izuzetno važan) deo aplikacije za razmenu e-pošte.

2.1.6 Mrežne aplikacije obuhvaćene ovom knjigom

Nove internet aplikacije javnog domena, kao i vlasničke, pojavljuju se skoro svakodnevno. Umesto da, poput neke enciklopedije, prikažemo veliki broj internet aplikacija, usredsredićemo se na manji broj važnijih i najčešće korišćenih aplikacija. U ovom poglavlju prikazaćemo pet izuzetno važnih aplikacija: veb, prenos datoteka, elektronsku poštu, servis imenovanja direktorijuma i P2P aplikacije. Počećemo od veba, ne samo zato što je to enormno popularna aplikacija, već i zato što je njen protokol aplikativnog sloja, HTTP, jednostavan i lako razumljiv. Pošto obradimo veb, ukratko ćemo objasniti protokol FTP (engl. File Transfer Protocol – protokol za prenos datoteka), koji je suprotnost protokolu HTTP. Zatim govorimo o elektronskoj pošti, prvoj izuzetno popularnoj internet aplikaciji. Aplikacije za elektronsku poštu mnogo su složenije od veb aplikacije po tome što ne koriste samo jedan protokol aplikativnog sloja, već nekoliko njih. Posle e-pošte, govorimo o sistemu

imenovanja domena (Domain Naming System, DNS) koji obezbeđuje uslugu imena za internet. Većina korisnika nema neposredan dodir sa sistemom imenovanja domena jer se ova usluga pokreće posredno preko drugih aplikacija (uključujući veb, prenos datoteka i elektronsku poštu). DNS na izuzetno lep način prikazuje kako se deo funkcionalnosti jezgra mreže (prevođenje imena na mreži u mrežnu adresu) primenjuje na aplikativnom sloju interneta. Konačno, govorićemo u ovom poglavlju o nekoliko P2P aplikacija, u koje spadaju aplikacije za razmenu datoteka i distribuirana pretraga. U poglavlju 7, obradićemo multimedijalne aplikacije, uključujući video u realnom vremenu i razgovor preko interneta.

2.2 Veb i HTTP

Sve do ranih devedesetih godina, internet su prvenstveno koristili istraživači, akademska zajednica i studenti za umrežavanje sa udaljenim računarima, prenos datoteka sa lokalnih na udaljene računare i obratno, prijem i slanje vesti, kao i za prijem i slanje elektronske pošte. Iako su ove aplikacije bile (i još uvek su) veoma korisne, internet je izvan ovih akademskih i istraživačkih zajednica, praktično bio nepoznat. Međutim, ranih devedesetih godina na scenu stupa značajna nova aplikacija – *World Wide Web* [Berners-Lee 1994]. Veb je prva internet aplikacija koja je privukla znatiželju široke javnosti. Ona je dramatično izmenila i nastavila da menja način na koji ljudi međusobno sarađuju unutar i izvan svog radnog okruženja. Veb je pomogao internetu da od samo jedne od mnogih postane jedina mreža za razmenu podataka.

Većinu korisnika najviše privlači to što veb radi *na zahtev*. Korisnici primaju ono što žele i onda kada to žele. Ovo se razlikuje od načina rada tradicionalnih radio i televizijskih stanica koje korisnike primoravaju da se „uključe" onda kada dobavljač sadržaja učini sadržaj raspoloživim. Pored toga što je dostupan na zahtev, veb ima i mnoge druge prednosti zbog kojih ga korisnici vole i cene. Svako može na krajnje jednostavan način da ponudi neku informaciju putem veba – svi mogu da postanu izdavači uz minimalne izdatke. Hiperveze i pretraživači olakšavaju snalaženje kroz mnoštvo veb lokacija. Grafički elementi utiču na naša čula. Obrasci, Java Scrip, Java apleti i brojne druge komponente, omogućavaju lak pristup veb stranicama i lokacijama. Veb služi i kao platforma za mnoge fantastične aplikacije koje su se pojavile posle 2003. godine, uključujući *YouTube*, *Gmail* i *Facebook*.

2.2.1 Prikaz protokola HTTP

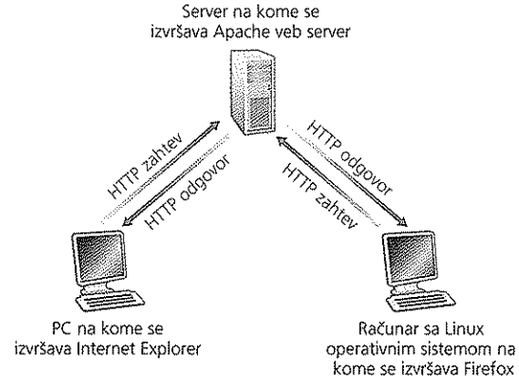
Srce veba je protokol **HTTP** (Hyper Text Transfer Protocol – protokol za prenos hiperteksta) koji se nalazi na njegovom aplikativnom sloju. HTTP je definisan dokumentima [RFC 1945] i [RFC 2616]. HTTP se implementira kroz dva programa: klijentski i serverskiprogram. Klijentski i serverski programi, koji se izvršavaju na dva različita krajnja sistema, komunicirajujedan sa drugimrazmenom HTTP poruka. Protokol HTTP definiše strukturu tih poruka i način na koji ih klijent i server razmenjuju. Pre nego što detaljno objasnimo protokol HTTP, prvo ćemo se bolje upoznati sa veb terminologijom.

Veb strana (ili dokument) sastoji se od objekata. **Objekat** je u suštini neka datoteka – kao što je HTML datoteka, JPEG slika, Java aplet ili video zapis – koju je moguće adresirati jedinstvenim uniformnim lokatorom resursa (Uniform Resource Locator, URL). Većina veb stranica sastoji se od **osnovne HTML datoteke** i nekoliko objekata koji se u njoj linkuju. Ukoliko, na primer, veb stranica sadrži HTML tekst i pet JPEG slika, onda ona ima šest objekata: osnovnu HTML datoteku i tih pet slika. Osnovna HTML datoteka linkuje ostale objekte na stranici putem njihovih URL adresa. Svaka URL adresa sastoji se od dva dela: naziva servera na kome se određeni objekat nalazi i putanje do objekta. Na primer, URL adresu:

```
http://www.someSchool.edu/someDepartment/picture.gif
```

sačinjava deo `www.someSchool.edu` za naziv računara i deo `/someDepartment/picture.gif` za putanju do objekta. Pošto se na **veb pretraživačima** (kao što su *Internet Explorer* ili *Firefox*) realizuje klijentska strana protokola HTTP, kada govorimo o vebu, ravnopravno koristimo izraze *veb pretraživač* i *klijent*. Na **veb serverima**, na kojima se realizuje serverska strana protokola HTTP, nalaze se veb objekti do kojih se dolazi navođenjem njihovih URL adresa. U popularne veb servere spadaju *Apache* i *Microsoft Internet Information Server*.

HTTP definiše način na koji veb klijenti traže veb strane od veb servera, kao i način na koji veb serveri šalju tražene strane klijentima. Kasnije ćemo posvetiti više pažnje interakciji između klijenta i servera, a osnovna ideja je prikazana na slici 2.6. Kada korisnik zatraži neku veb stranicu (na primer, pritiskom miša na neku hipervezu), pretraživač šalje serveru HTTP poruke zahteva za objekte sa odgovarajuće stranice. Server prima ove zahteve i odgovara HTTP porukama odgovora u kojima se nalaze traženi objekti.



Slika 2.6 ◆ Slanje zahteva i dobijanje odgovora u protokolu HTPP

HTTP koristi TCP kao svoj osnovni transportni protokol (i nikad ne koristi protokol UDP). HTTP klijent najpre uspostavlja TCP vezu sa serverom. Pošto se veza uspostavi, procesi veb pretraživača i servera pristupaju TCP protokolu kroz interfejse svojih soketa. Kao što smo rekli u odeljku 2.1, na klijentskoj strani interfejs

soketa predstavlja vrata između klijentskog procesa i TCP veze; na serverskoj strani su to vrata između serverskog procesa i TCP veze. Klijent šalje HTTP poruke zahteva kroz interfejs soketa i iz interfejsa svog soketa prima HTTP poruke odgovora. Slično tome, HTTP server iz interfejsa svog soketa prima poruke zahteva, a poruke odgovora šalje kroz interfejs svog soketa. Onog trenutka kada klijent pošalje neku poruku kroz interfejs soketa, ta poruka više nije u njegovim rukama, već „u rukama" protokola TCP. Sećate se iz odeljka 2.1 da protokol TCP nudi uslugu pouzdanog prenosa podataka protokolu HTTP. To znači da svaka HTTP poruka zahteva, koju pošalje klijentskiproces, pre ili kasnije stiže neizmenjena do servera; isto tako, svaka HTTP poruka odgovora koju pošalje serverski proces pre ili kasnije neizmenjena stiže do klijenta. Ovde vidimo jednu od velikih prednosti slojevite arhitekture – protokol HTTP ne mora da brine o gubitku podataka i o tome kako protokol TCP obnavlja izgubljene podatke, ili kako menja redosled podataka unutar mreže. To je zadatak protokola TCP i protokola na nižim slojevima skupa protokola.

Veoma je važno naglasiti da server šalje tražene datoteke klijentu bez čuvanja bilo kakvih podataka o stanju klijenta. Ukoliko neki klijent u roku od nekoliko sekundi dvaput zatraži isti objekat, server ne odgovara tako što kaže da je klijentu upravo poslao određeni objekat, već ponovo šalje traženi objekat, kao da je zaboravio da je to već jednom učinio. Pošto HTTP server ne održava nikakve informacije o klijentima, za protokol HTTP se kaže da je **protokol bez stanja**. Takođe, kao što smo rekli u odeljku 2.1, podsećamo da veb koristi klijentsko-serversku arhitekturu. Veb server je uvek dostupan, ima nepro-menljivu IP adresu i uslužuje zahteve ponekad i više miliona različitih veb pretraživača.

2.2.2 Nepostojane i postojane veze

Kod mnogih internet aplikacija, klijent i server komuniciraju duže vreme, pri čemu klijent šalje niz zahteva, a server odgovara na sve te zahteve. U zavisnosti od aplikacije i od toga kako se aplikacija koristi, nizovi zahteva mogu da se šalju jedan za drugim, periodično u pravilnim intervalima ili povremeno. Kada se interakcija između klijenta i servera uspostavlja preko protokola TCP, programer aplikacije mora da donese značajnu odluku – da li se svaki par zahteva i odgovora šalje preko *zasebne* TCP veze, ili se svi zahtevi odgovori na njih šalju preko *iste* TCP veze? U prvom slučaju, za aplikaciju se kaže da koristi **nepostojane veze**, a u potonjem slučaju **postojane veze**. Da bismo bolje razumeli ovaj problem pri projektovanju aplikacije, istražićemo prednosti i mane postojanih veza za neku određenu aplikaciju, u ovom slučaju HTTP, koja može da koristi obe vrste veza. Mada HTTP u podrazumevanom režimu koristi postojanu vezu, HTTP klijenti i serveri mogu da se konfigurišu tako da koriste i nepostojanu vezu.

HTTP sa nepostojanim vezama

Prođimo sada kroz postupak prenošenja neke veb stranice od servera do klijenta u slučaju nepostojanih veza. Pretpostavimo da se ta stranica sastoji od osnovne HTML datoteke i deset JPEG slika i da se svih 11 objekata nalazi na istom serveru. Pretpostavićemo i da je URL adresa osnovne HTML datoteke

```
http://www.someSchool.edu/someDepartment/home.gif
```

Evo šta se dešava:

1. HTTP klijentski proces uspostavlja TCP vezu sa serverom www.someSchool.edu na portu broj 80, što je podrazumevani broj porta za protokol HTTP. Ovoj TCP vezi pridružuju se soket na klijentu i soket na serveru.
2. HTTP klijent šalje HTTP poruku zahteva serveru preko svog soketa. Poruka zahteva sadrži naziv putanje `/someDepartment/home.home.index`. (Uskoro ćemo podrobnije govoriti o HTTP porukama.)
3. HTTP serverski proces prima poruku zahteva preko svog soketa, učitava objekat `/someDepartment/home.home.index` iz svoje memorije (RAM ili disk), enkapsulira objekat u HTTP poruku odgovora i tu poruku šalje klijentu preko svog soketa.
4. HTTP proces servera saopštava protokolu TCP da prekineTCP vezu. (Mada protokol TCP u stvari ne prekida ovu vezu sve dok se sasvim ne uveri da je klijent primio neizmenjenu poruku odgovora.)
5. HTTP klijent prima poruku odgovora. TCP veza se prekida. U poruci je naznačeno da je enkapsulirani objekat HTML datoteka. Klijent izvlači tu datoteku iz poruke, ispituje HTML datoteku i pronalazi reference do deset JPEG objekata.
6. Prva četiri koraka se zatim ponavljaju za sve navedene JPEG objekte.

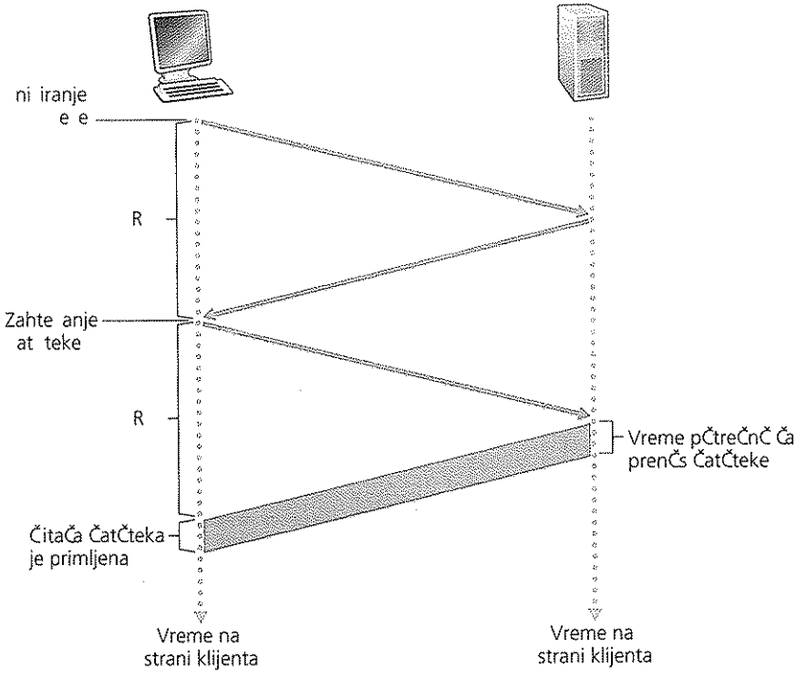
Kada veb pretraživač primi određenu veb stranicu on je prikazuje korisniku. Dva različita veb pretraživača mogu da istu veb stranicu protumače (odnosno prikažu korisniku) donekle drugačije. HTTP nema ništa sa tim kako klijent prikazuje veb stranice. HTTP specifikacije ([RFC 1945] i [RFC 2616]) definišu samo komunikacioni protokol između klijentskog HTTP programa i serverskog HTTP programa.

Navedenim koracima prikazano je korišćenje nepostojanih veza pri čemu se svaka TCP veza prekida pošto server pošalje određeni objekat – ta veza ne postoji za ostale objekte. Obratite pažnju na to da svaka TCP veza prenosi samo jednu poruku zahteva i jednu poruku odgovora. Prema tome, u ovom primeru, kada korisnik zatraži odgovarajuću veb stranicu, uspostavlja se 11 TCP veza.

U navedenim koracima namerno nismo naveli da li je klijent svih deset JPEG slika dobio putem deset uzastopnih TCP veza, ili je neke od njih dobio i putem paralelnih TCP veza. U stvari, korisnici mogu da konfigurišu savremene veb pretraživače tako da kontrolišu stepen paralelizma. U podrazumevanim režimima većina veb pretraživača otvara pet do deset paralelnih TCP veza i svakoj od tih veza obrađuje

se po jedna transakcija zahtev-odgovor. Ukoliko korisnik to želi, on može da ograniči broj paralelnih veza na samo jednu i u tom slučaju bilo bi uspostavljeno deset uzastopnih veza. Kao što ćemo videti u sledećem poglavlju, korišćenje paralelnih veza ubrzava vreme odziva.

Pre nego što nastavimo, pokušaćemo da izračunamo koliko vremena prođe od trenutka kada klijent zatraži osnovnu HTML datoteku do trenutka kada klijent primi čitavu datoteku. Do sada smo **vreme povratnog puta** (round-trip time, **RTT**) definisali kao vreme koje je potrebno malom paketu za put od klijenta do servera, a zatim nazad do klijenta. RTT obuhvata kašnjenja usled prostiranja paketa, kašnjenja usled čekanja u redu u ruterima i komutatorimana putanji, kao i kašnjenja usled obrade paketa. (Ova kašnjenja obradili smo u odeljku 1.4). Sada razmotrimo šta se dešava kada korisnik izabere neku hipervezu. Kao što je prikazano na slici 2.7, veb pretraživač inicira TCP vezu sa veb serverom; što obuhvata „trostruko rukovanje" – klijent šalje mali TCP segment serveru, server potvrđuje prijem i odgovara malim TCP segmentom i, konačno, klijent šalje serveru potvrdu prijema. JednoRTT vreme obuhvata prva dva dela ovog „trostrukog rukovanja". Pošto se završe prva dva dela rukovanja, klijent uspostavljenom TCP vezom šalje HTTP poruku zahteva zajedno sa trećim delom „trostrukog rukovanja" (potvrdom prijema). Kada poruka zahteva



Slika 2.7 ◆ Izračunavanje vremena neophodnog za slanje zahteva i prijem HTML datoteke

stigne do servera, on uspostavljenom TCP vezom šalje odgovarajuću HTML datoteku. Ovi HTTP zahtev i odgovor troše još jednoRTT vreme. Dakle, grubo rečeno, ukupno vreme odziva dobija se sabiranjem dva RTT vremena i trajanja prenosa HTML datoteke od servera do klijenta.

HTTP sa postojanim vezama

Nepostojane veze imaju svoje nedostatke. Najpre, za *svaki traženi objekat* mora da se uspostavi i održava potpuno nova veza. Svim tim vezama mora da se dodeli prihvatna TCP memorija (TCP bafer), a klijent i server moraju da održavaju parametre svih TCP veza. Sve ovo može značajno da optereti veb server koji istovremeno ispunjava zahteve nekoliko stotina različitih klijenata. Zatim, kao što smo upravo opisali, isporuka svih objekata kasni dva RTT vremena – jedan za uspostavljanje TCP veze, a drugi za slanje zahteva i prijem objekta.

Kod postojanih veza, nakon slanja odgovora server ostavlja TCP vezu otvorenom. Naknadni zahtevi i odgovori između istog klijenta i servera mogu da se pošalju tom istom vezom. Drugim rečima, čitava veb stranica (u prethodnom primeru, osnovna HTML datoteka i deset slika) može se poslati jednom postojanom vezom. Štaviše, jednom postojanom TCP vezom moguće je poslati istom klijentu više veb stranica sa istog veb servera. Zahteve za objekte moguće je slati jedan za drugim, bez čekanja na odgovor za prethodno poslate zahteve (paralelna obrada). HTTP server obično prekida ovakvu vezu ukoliko se ne koristi neko određeno vreme (vremenski interval koji je moguće podesiti). Kako server prima zahteve jedan za drugim, tako i šalje objekte. U podrazumevanom režimu rada HTTP protokola koristi se postojana veza sa paralelnom obradom. Kvantitativno ćemo uporediti performanse nepostojanih i postojanih veza u domaćim zadacima u poglavljima 2 i 3. Predlažemo da pogledate i sledeće radove autora [Heidemann 1997; Nielsen 1997].

2.2.3 Format HTTP poruke

Specifikacije protokola HTTP [RFC 1945; RFC 2616] obuhvataju i definicije formata HTTP poruka. Postoje dve vrste HTTP poruka, poruke zahteva i poruke odgovora, i o njima govorimo u produžetku.

HTTP poruka zahteva

Evo kako izgleda uobičajena HTTP poruka zahteva:

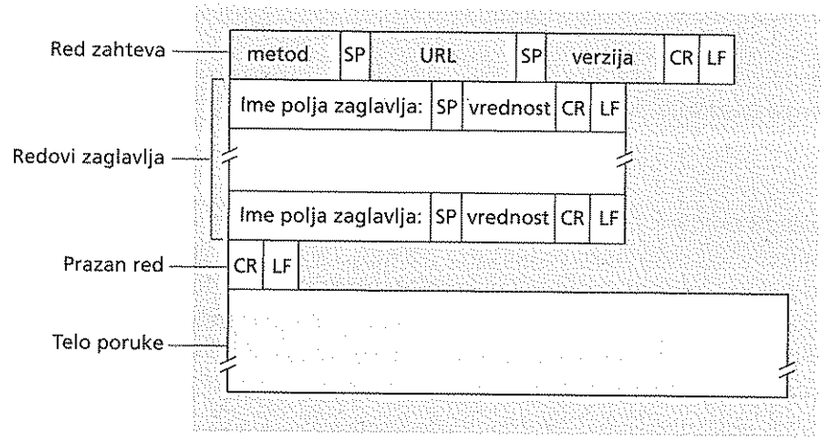
```
GET/somedir/page.htmlHTTP/1.1
Host:www.nekaskola.edu
```

```
Connection:close
User-agent:Mozilla/5.0
Accept-language:fr
```

Mnogo toga možemo da naučimo pažljivim posmatranjem ove jednostavne poruke zahteva. Pre svega, vidimo da je napisana kao običan ASCII tekst, tako da prosečno računarski pismenaosoba može da je pročita. Zatim, vidimo da se sastoji od pet redova iza kojih dolazi znak za povratak na početak reda i znak za novi red. Iza poslednjeg reda nalazi se još jedan znak za povratak na početak reda i za novi red. Iako ova poruka zahteva ima pet redova, poruke zahtevamogu da imaju mnogo više redova, ali i samo jedan red. Prvi red HTTP poruke zahtevanaziva se **red zahteva**; svi ostali redovi su **redovi zaglavlja**. Red zahteva ima tri polja: polje metoda, URL polje i polje HTTP verzije. Polje metoda može da ima više različitih vrednosti, uključujući `GET`, `POST`, `HEAD`, `PUT` i `DELETE`. HTTP poruke zahteva najčešće koriste metodu `GET`. Ova metoda se koristi kada veb pretraživač traži objekat, pri čemu se zahtevani objekat navodi u URL polju. U ovom primeru, veb čitač zahteva objekat `/somedir/page.html`. Polje verzije jasno je samo po sebi; u ovom primeru, pretraživač koristi verziju protokola HTTP/1.1.

Pogledajmo sada redove zaglavlja u ovom primeru. Red zaglavlja `Host:www.nekaskola.edu` određuje računar na kome se traženi objekat nalazi. Možda mislite da je ovaj red zaglavlja suvišan, budući da je već uspostavljena TCP veza sa tim računarom. Međutim, kao što ćemo videti u odeljku 2.2.5, informacije koje pruža red zaglavlja host su neophodne zbog keš memorija proksi servera. Uključivanjem reda zaglavlja `Connection:close`veb pretraživač kaže serveru da ne želi da koristi postojane veze, već želi da server prekine vezu čim pošalje traženi objekat. U redu zaglavlja `User-agent:` naveden je korisnički agent, odnosno vrsta veb pretraživača koji je poslao zahtev serveru. U ovom slučaju to je program *Mozilla*/5.0, *Firefox* veb pretraživač. Ovaj red zaglavlja je koristan zato što omogućava serveru da različitim tipovima korisničkih agenata pošalje različite verzije istog objekta. (Sve te verzije imaju istu URL adresu.) Konačno, u redu zaglavlja `Accept-language:` navodi se da bi korisnik voleo da primi francusku verziju objekta, ukoliko postoji na serveru; u suprotnom, server šalje podrazumevanu verziju traženog objekta. Red zaglavlja `Accept-language:` samo je jedan od mnogih kojima se u okviru protokola HTTP pregovara o sadržaju.

Pošto smo završili sa primerom, pogledajmo kako izgleda opšti format poruke zahteva, prikazane na slici 2.8. Vidimo da opšti format strogo sledi naš prethodni primer. Ipak, možda ste primetili da nakon redova zaglavlja (i dodatnih znakova povratak na početak reda i za novi red) sledi „telo poruke". Telo poruke je prazno kod metoda `GET`, ali se koristi kod metoda `POST`. Kada korisnik ispunjava neki obrazac, HTTP klijenti obično koriste metod `POST`– na primer, kada u odgovarajuće polje pretraživača upisuje reči za pretragu. `POST` porukom korisnik i dalje od veb servera zahteva određenu veb stranicu, ali tačan sadržaj te stranice zavisi od onoga što je korisnik upisao u poljima obrasca. Ako je vrednost polja metoda POST, onda se u telu poruke nalazi ono što je korisnik upisao u poljima obrasca.



Legenda:
SP - Razmak
CR - Znak za novi red
LF - Znak za povratak na početak reda

Slika 2.8 ◆ Opšti format HTTP poruke zahteva

Bili bismo zbilja nemarni kada ne bismo spomenuli i to da zahtev koji nastaje popunjavanjem obrasca ne mora uvek da koristi metod `POST`. HTML obrasci često koriste i metod `GET` i unete podatke (u polja obrasca) unose u zahtevanu URL adresu. Na primer, kada obrazac koristi metod `GET` i ima dva polja u koja je upisano `majmuni` i `banane`, onda će URL izgledati poput `www.nekavebstrana.com/pretraga?majmuni&banane`. U svakodnevnom krstarenju vebom verovatno ste već mnogo puta videli ovako proširenu URL.

Metod `HEAD` sličan je metodu `GET`. Kada primi zahtev sa metodom `HEAD`, server odgovara HTTP porukom, ali izostavlja traženi objekat. Programeri aplikacija često koriste ovaj metod za otklanjanje grešaka u programima. Metod `PUT` se obično koristi zajedno sa alatkama za objavljivanje dokumenata na vebu. On korisniku omogućava da neki objekat prebaci na tačno određenu putanju (direktorijum) određenog veb servera. Takođe, metod `PUT` koriste i aplikacije kojima je neophodno da prenesu objekte na veb server. Metod `DELETE` omogućava korisniku ili aplikaciji da obriše neki objekat sa veb servera.

HTTP poruka odgovora

U nastavku je data uobičajena HTTP poruka odgovora. Ova poruka mogla bi da predstavlja odgovor na poruku zahteva o kojoj smo prethodno govorili.

```
HTTP/1.1 200 OK
Connection: close
Date: Thu, 09Aug 201115:44:04 GMT
```

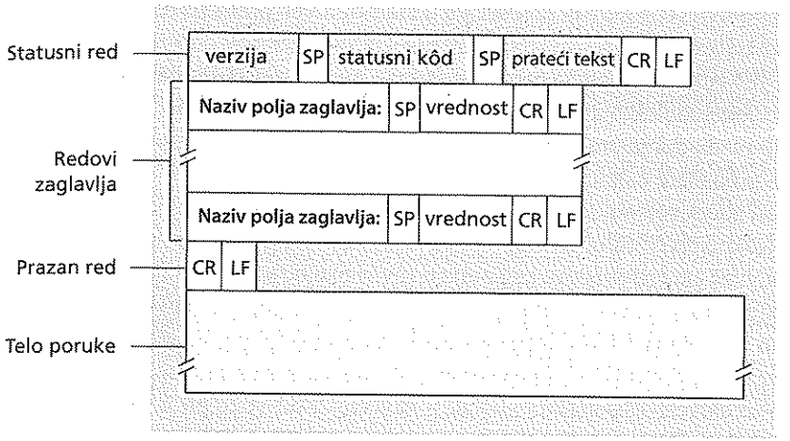
```
Server: Apache/2.2.3 (CentOS)
Last-Modified: Tue, 09Aug 201115:11:03 GMT
Content-Length: 6821
Content-Type: text/html

(podaci podaci podaci podaci podaci ...)
```

Pogledajmo malo pažljivije ovu poruku odgovora. Ona ima tri dela: početni **statusni red**, šest **redova zaglavlja** i **telo poruke**. Telo poruke predstavlja najvažniji deo poruke – ovde se nalazi traženi objekat (predstavljeni redom `podaci podaci podaci podaci podaci ...`). Statusni red ima tri polja: polje verzije protokola, statusni kôd i odgovarajuću statusnu poruku. U ovom primeru statusnim redom navedeno je da server koristi protokol HTTP/1.1 i da je sve u redu (`OK`), odnosno da je server pronašao traženi objekat i da ga šalje.

Pogledajmo sada redove zaglavlja. Server koristi red zaglavlja `Connection: close` kako bi saopštio klijentu da namerava da,nakon što pošalje traženi objekat prekine tu TCP vezu. U redu zaglavlja `Date:` navedeni su datum i vreme kada je server napravio i poslao HTTP odgovor. Skrećemo vam pažnju da to nije vreme kada je traženi objekat napravljen ili poslednji put promenjen, već vreme kada ga je server učitao iz svog sistema datoteka, stavio u poruku odgovora i poslao poruku odgovora. Red zaglavlja `Server:` naznačava da je poruku generisao veb server *Apache*; ovaj red odgovara redu zaglavlja `User-agent:` iz HTTP poruke zahteva. U redu zaglavlja `Last-Modified:` navodi se vreme i datum kada je traženi objekat napravljen ili poslednji put izmenjen. Zaglavlje`Last-Modified:`, kome ćemo uskoro posvetiti nešto više pažnje, izuzetno je važan za keširanje objekta – kako na lokalnom klijentu, tako i na mrežnim serverima za keširanje (oni se nazivaju i proksi serveri). U redu zaglavlju `Content-Length:` navodi se broj bajtova u objektu koji se šalje u telu poruke. Red zaglavlja `Content-Type:` označava da je objekat koji se nalazi u telu poruke HTML tekst. (Tip objekta se zvanično označava redom zaglavlja `Content-Type:` a ne oznakom tipa datoteke.)

Pošto smo pregledali ovaj primer, pogledajmo kako izgleda opšti format poruke odgovora, prikazan na slici 2.9. Opšti format poruke odgovora poklapa se sa prethodnim primerom poruke odgovora. Recimo nešto i o statusnim kodovima i pratećem tekstu. Statusnim kodom i pratećim tekstom navodi se rezultat zahteva. Evo nekih uobičajenih statusnih kodova i odgovarajućih pratećih tekstova:

- `200 OK`: zahtev je uspešno izvršen i zahtevana informacija se vraća u odgovoru.
- `301 Moved Permanently`: traženi objekat je trajno uklonjen; novi URL se navodi u zaglavlju `Location:` poruke odgovora; klijentski softver tako automatski dolazi do nove URL adrese.
- `400 Bad Request`: opšta oznaka greške koja naznačava da server nije razumeo zahtev.



Slika 2.9 ◆ Opšti format HTTP poruke odgovora

- `404 Not Found`: traženi dokument ne postoji na ovom serveru.
- `505 HTTP Version Not Supported`: server ne podržava traženu verziju protokola HTTP.

Video napomena

rišćenje pr grama Wireshark a ispiti anje H pr t k la

Da li biste želeli da vidite pravu HTTP poruku odgovora? Osim što je veoma preporučljivo, to je i krajnje jednostavno. Prvo uspostavite Telnet vezu sa svojim omiljenim veb serverom. Zatim upišite poruku zahteva od jednog reda kojom tražite neki objekat koji se nalazi na serveru. Na primer, u komandnu liniju upišite:

```
telnet cis.poly.edu 80

GET /~ross/ HTTP/1.1
Host: cis.poly.edu
```

(Pošto uneste poslednji red, dva puta pritisnite taster Enter.) Time se uspostavlja TCP veza preko porta 80 računara `cis.poly.edu` i zatim šalje ova HTTP poruka zahteva. Trebalo bi da vidite poruku odgovora i u okviru nje osnovnu HTML datoteku početne strane prezentacije profesora Rosa. Ukoliko biste, umesto samog objekta, više voleli da vidite redove HTTP poruke, metod `GET` zamenite metodom `HEAD`. Konačno, zamenite `/~ross/` sa `/~banana/` i pogledajte kako će izgledati odgovor.

U ovom odeljku govorili smo o više vrsta redova zaglavlja koji mogu da se koriste u HTTP porukama zahteva i odgovora. Specifikacija protokola HTTP definiše

brojne druge redove zaglavlja koje mogu da ubacuju veb pretraživači, veb serveri i mrežni serveri za keširanje. Mi smo ovde prikazali samo manji deo. Vrlo brzo ćemo pomenuti još neke redove zaglavlja, dok ćemo o nekim drugim govoriti na kraju poglavlja u odeljku 2.2.5 koji je posvećen mrežnom keširanju veba. Iscrpno i lako razumljivo razmatranje protokola HTTP, uključujući i zaglavlja i statusne kodove, možete da pronađete u delu [Krishnamurty 2001].

Na koji način veb pretraživač odlučuje koji će se redovi zaglavlja naći u poruci zahteva? Kako server odlučuje koji će se redovi zaglavlja naći u poruci odgovora? Veb pretraživač generiše redove zaglavlja u zavisnosti od vrste i verzije pretraživača (na primer, pretraživač verzije HTTP/1.0 neće generisati redove zaglavlja za verziju 1.1), zatim od načina na koji ga je korisnik podesio (recimo, željeni jezik), kao i od toga da li veb pretraživač ima keširanu, ali možda zastarelu, verziju određenog objekta. Veb serveri se ponašaju slično: u zavisnosti od vrste proizvoda, verzije i konfiguracije, razlikovaće se i redovi zaglavlja koji se nalaze u porukama odgovora.

2.2.4 Interakcija između korisnika i servera: kolačići

Nešto ranije napomenuli smo da HTTP serveri rade bez održavanja stanja. Ovo pojednostavljuje njihovo projektovanje i omogućava inženjerima da razviju moćne veb servere koji mogu da upravljaju hiljadama istovremenih TCP veza. Ipak, ponekad je veb stranicama potrebno da identifikuju korisnike, bilo zato što server želi da ograniči pristup, bilo zato što sadržaj koji se nudi zavisi od identiteta korisnika. Protokol HTTP u tu svrhu koristi kolačiće (eng. cookies). Kolačići, definisani dokumentom [RFC 6265], omogućavaju veb lokacijama da čuvaju podatke o korisnicima. Većina važnijih komercijalnih veb lokacija danas koristi kolačiće.

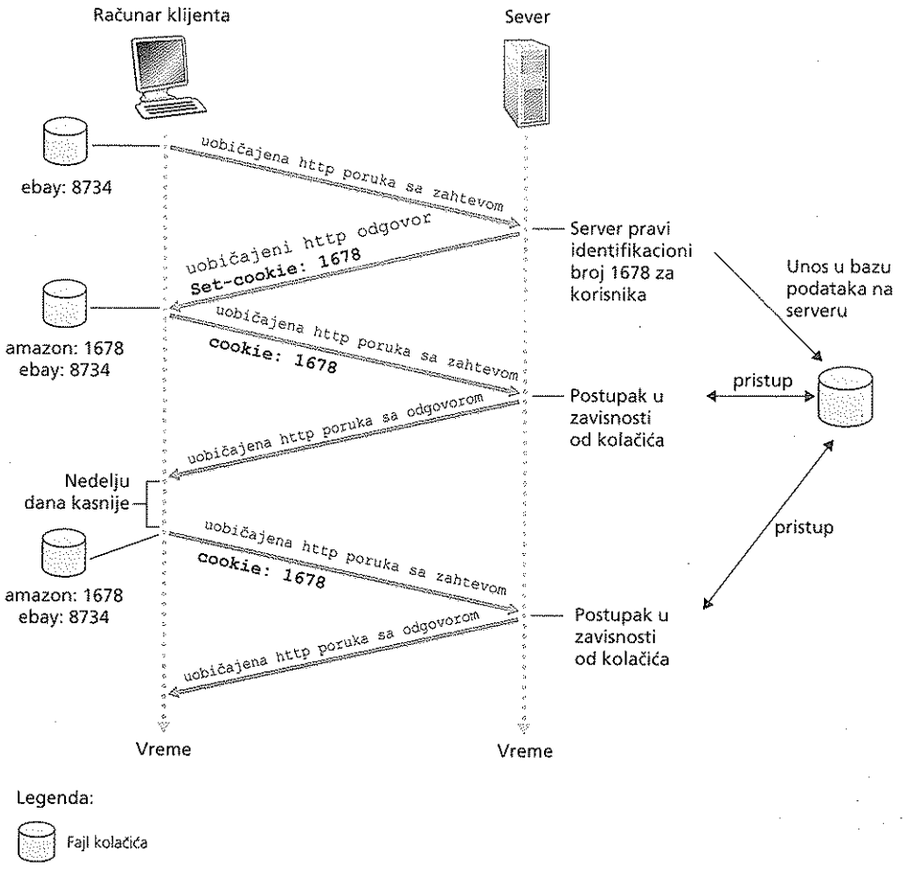
Kao što je prikazano na slici 2.10, tehnologija kolačića ima četiri komponente: (1) red zaglavlja kolačića u HTTP poruci odgovora; (2) red zaglavlja kolačića u HTTP poruci zahteva; (3) datoteku kolačića koja se čuva na krajnjem sistemu korisnika, a kojim upravlja njegov veb pretraživač i (4) pozadinsku bazu podataka veb lokacije. Koristeći sliku 2.10, razmotrimo primer načina na koji kolačići rade. Pretpostavimo da je Suzana, koja vebu pristupa koristeći *Internet Explorer* na svom kućnom računaru, prvi put posetila lokaciju Amazon.com. Pretpostavimo da je ranije već posećivala lokaciju eBay. Kada njen zahtev stigne na veb server lokacije Amazon, taj server pravi jedinstveni identifikacioni broj, kao i zapis u bazi podataka na tom serveru koja se indeksira tim identifikacionim brojem. Veb server lokacije Amazon zatim Suzaninom pretraživaču šalje HTTP odgovor u kome se nalazi i zaglavlje `Set-cookie:` koje sadrži odgovarajući identifikacioni broj. Na primer, ovaj red zaglavlja mogao bi da izgleda ovako:

```
Set-cookie: 1678
```

Kada Suzanin pretraživač primi ovu HTTP poruku odgovora, on će ugledati i zaglavlje `Set-cookie:`. Pretraživač zatim dodaje red u posebnu datoteku kolačića kojom on upravlja. U ovaj red upisuju se ime računara servera i identifikacioni

broj iz zaglavlja `Set-cookie:`. Imajte na umu da ova datoteka kolačića već ima unos za lokaciju eBay, pošto je Suzana ranije posećivala tu lokaciju. Dok Suzana bude pretraživala lokaciju Amazon, svaki put kada zatraži neku veb stranicu, njen pretraživač će pogledati datoteku kolačića, izvući će njen identifikacioni broj za tu veb lokaciju i staviti ga u red zaglavlja kolačića HTTP zahtevu. Drugim rečima, svaki njen HTTP zahtev upućen serveru lokacije Amazon obuhvata red zaglavlja:

```
Cookie: 1678
```



Slika 1.10 ◆ Vođenje podataka o korisnicima korišćenjem kolačića

Na ovaj način server lokacije Amazon može da prati šta Suzana radi na lokaciji Amazon. Iako ova veb lokacija ne zna Suzanino ime, ona tačno zna koje je stranice posetio korisnik broj 1678, kojim redom i u koje vreme! Amazon koristi kolačiće kako bi olakšao kupovinu na svojoj lokaciji (eng. shopping card) – Amazon vodi računa o svemu što Suzan želi da kupi, tako da ona, kada završi kupovinu, može zajedno da ih plati .

Ukoliko se Suzana, na primer, nedelju dana kasnije vrati na ovu lokaciju, njen pretraživač će nastaviti da u poruke zahteva stavlja red zaglavlja `Cookie:1678`. U zavisnosti od toga koje je stranice Suzana ranije posećivala na lokaciji Amazon, ova lokacija može da joj preporuči određene proizvode. Ukoliko Suzan odluči da se na ovoj lokaciji i registruje – odnosno da upiše puno ime i prezime, e-adresu, običnu poštansku adresu i informacije o kreditnoj kartici – Amazon će ove informacije uneti u svoju bazu podataka i tako Suzanino ime povezati sa njenim identifikacionim brojem (i svim stranama koje je u prošlosti posetila na ovoj lokaciji!). Upravo na ovaj način Amazon i ostale lokacije za elektronsku trgovinu obezbeđuju „kupovinu jednim potezom mišem" – kada Suzana pri nekoj narednoj poseti odluči da nešto kupi, neće morati ponovo da upisuje svoje ime, broj kreditne kartice i adresu.

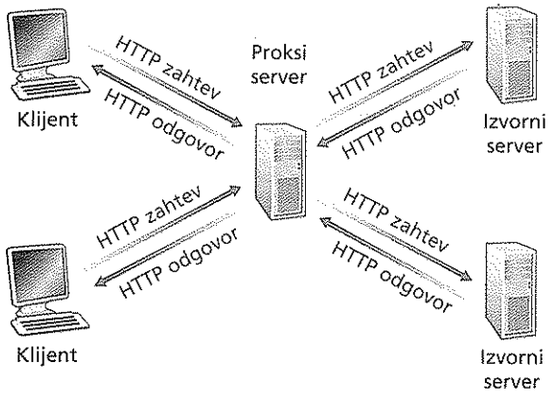
Iz ovoga možemo da zaključimo da se kolačići mogu koristiti za identifikaciju korisnika. Kada prvi put poseti neku lokaciju, korisnik može da se identifikuje (recimo, navođenjem njegovog ili njenog imena). Posle toga pretraživač prilikom svake naredne sesije prosleđuje zaglavlje kolačića serveru i tako identifikuje korisnika serveru. Stoga se kolačići mogu koristiti za pravljenje sloja sesije korisnika preko protokola HTTP koji radi bez održavanja stanja. Kada se, na primer, korisnik prijavi na veb aplikaciju za e-poštu (kao što je *Hotmail*), pretraživač šalje serveru informaciju iz kolačića, omogućavajući serveru da sve vreme dok korisnik koristi tu aplikaciju, zna o kom korisniku se radi.

Iako značajno pojednostavljuju kupovinu preko interneta, kolačići su istovremeno i veoma kontroverzni zato što na neki način zadiru u privatnost korisnika. Kao što smo upravo videli, kombinovanjem informacija iz kolačića sa podacima koje korisnik dostavi kroz korisnički nalog, veb lokacija može da sazna dosta toga o korisniku i da svoja saznanja možda proda nekom trećem. Dokument *Cookie Central* [Cookie Central 2012] sadrži dosta informacija o kontroverznim stranama kolačića.

2.2.5 Veb keširanje

Server za veb keširanje – takođe se naziva i **proksi server** – sastavni je deo mreže koji ispunjava HTTP zahteve u ime veb servera na kome se nalazi izvorni objekat. Veb keš ima sopstvene diskove za skladištenje na kojima čuva kopije nedavno traženih objekata. Kao što je prikazano na slici 2.11, korisnikov pretraživač može da se konfiguriše tako da se svi korisnikovi HTTP zahtevi najpre usmeravaju ka veb keš. Kada se pretraživač tako konfiguriše, svi zahtevi tog pretraživača prvo se usmeravaju na veb keš. Primera radi, pretpostavimo da pretraživač zahteva objekat `http://www.nekaskola.edu/zgrada.gif`. Evo šta se dešava:

1. Veb čitač uspostavlja TCP vezu sa serverom za veb keširanje i šalje mu HTTP poruku zahteva za određeni objekat.



Slika 2.11 ◆ Klijenti zahtevaju objekte, koristeći server za veb keširanje

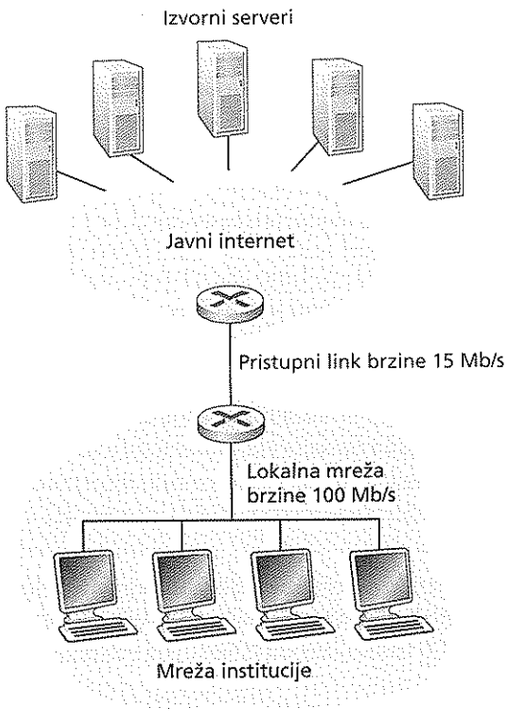
2. Veb keš server proverava da li poseduje lokalno uskladištenu kopiju tog objekta. Ukoliko je ima, server za veb keširanje vraća klijentovom pretraživaču odgovarajući objekat unutar HTTP poruke odgovora.
3. Ukoliko ne poseduje traženi objekat, server za veb keširanje uspostavlja TCP vezu sa izvornim serverom, u ovom slučaju, sa serverom `www.nekaskola.edu`. Server za veb keširanje šalje HTTP zahtev za određeni objekat uspostavljenom TCP vezom između njega i odgovarajućeg servera. Pošto primi ovaj zahtev, izvorni server šalje traženi objekat u okviru HTTP odgovora serveru za veb keširanje.
4. Kada server za veb keširanje primi ovaj objekat, skladišti njegovu kopiju na svom disku i kopiju prosleđuje klijentovom pretraživaču u okviru HTTP poruke odgovora(postojećom TCP vezom između klijentovog veb pretraživača i severa za veb keširanje).

Skrećemo vam pažnju na to da je server za veb keširanje istovremeno i server i klijent. Kada prima zahteve od klijentovog pretraživača i šalje mu odgovore, ima ulogu servera. Kada šalje zahteve izvornom serveru i prima odgovore od njega, on je klijent.

Server za veb keširanje obično kupuje i postavlja posrednik za internet usluge. Na primer, univerzitet može da instalira server za veb keširanje u svojoj mreži nakon čega se svi veb pretraživači u okviru kampusa konfigurišu tako da budu usmereni prema njemu. Isto tako, veliki oblasni posrednik za internet usluge (kao što je AOL) može da u svojoj mreži instalira jedan ili više ovakvih servera i da zatim veb pretraživače koje isporučuje unapred konfiguriše tako da koriste postavljene servere za veb keširanje.

Postoje dva razloga zbog kojih se veb keširanje koristi na internetu. Najpre, veb keširanje značajno skraćuje vreme odziva na zahtev klijenta, naročito ukoliko je propusni opseg između klijenata i izvornih servera usko grlo u poređenju sa propusnim opsegom između klijenta i servera za veb keširanje. Ako između klijenta i servera za veb keširanje postoji veza velike propusne moći, a često je tako, i ukoliko se traženi objekat već nalazi u memoriji servera za veb keširanje, tada server za veb

keširanje može vrlo brzo da isporuči klijentu objekat. Zatim, kao što ćete uskoro videti u našem primeru, veb keširanje značajno smanjuje intenzitet saobraćaja na pristupnom linku neke institucije prema internetu. Smanjivanje saobraćaja znači da te institucije (univerziteta ili kompanije) ne moraju stalno da proširuju propusni opseg, čime se smanjuju troškovi. Štaviše, veb keširanje značajno smanjuje saobraćaja na čitavom internetu, čime se poboljšavaju performanse svih aplikacija.



Slika 2.12 ◆ Usko grlo između mreže neke institucije i interneta

Da biste bolje shvatili prednosti veb keširanja, evo jednog primera prikazanog na slici 2.12. Na ovoj slici prikazane su dve mreže – mreža neke institucije i ostatak javnog internet. Ta institucija koristi lokalnu mrežu velike brzine. Ruter ove mreže i ruter u okviru interneta su povezani linkom brzine 15 Mb/s. Izvorni serveri su povezani sa internetom, a nalaze se širom sveta. Pretpostavimo da je veličina prosečnog objekta 1 Mb i da veb pretraživači koji se koriste u instituciji svake sekunde upute prosečno 15 zahteva izvornim serverima. Pretpostavićemo i da su HTTP poruke zanemarljivo male tako da ne prave nikakav saobraćaj u mrežama ili na pristupnom linku (od rutera u instituciji do rutera u okviru interneta). Konačno, pretpostavićemo i to da od trenutka kada ruter na strani pristupnog linka ka internetu na slici 2.12 prosledi HTTP zahtev (u okviru IP datagrama) do trenutka kada dobije odgovor (obično kroz više IP datagrama) prođu prosečno dve sekunde. Radi lakšeg razumevanja, ovo poslednje kašnjenje ćemo nazivati „internet kašnjenje".

Ukupno vreme odziva – vreme koje protekne od trenutka kada veb pretraživač pošalje zahtev za objekat do trenutka kada primi taj objekat – predstavlja zbir kašnjenja u lokalnoj mreži, kašnjenja pristupnog linka (to jest, kašnjenje između dva rutera) i kašnjenje interneta. Sada ćemo napraviti grubu procenu ovog kašnjenja. Intenzitet saobraćaja u lokalnoj mreži (videti odeljak 1.4.2) iznosi:

$$(15 \text{ zahteva/sekundi}) \cdot (1 \text{ Mb/zahtev})/(100 \text{ Mb/s}) = 0{,}15$$

pri čemu je intenzitet saobraćaja napristupnom linku (od rutera na internetu do rutera institucije):

$$(15 \text{ zahteva/sekundi}) \cdot (100 \text{ Mb/zahtev})/(15 \text{ Mb/s}) = 1$$

Intenzitet saobraćaja od 0,15 u lokalnoj mreži znači čekanje koje se, u najgorem slučaju, meri desetinama milisekundi; prema tome kašnjenje u lokalnoj mreži možemo da zanemarimo. Međutim, kao što smo rekli u odeljku 1.4.2, kako se intenzitet saobraćaja približava 1 (kao što je to slučaj na pristupnom linku na slici 2.12), kašnjenje postaje sve duže i povećava se neograničeno. To znači da će se prosečno vreme odziva na zahteve korisnika meriti minutima, ako ne i duže, što je neprihvatljivo za korisnike ove institucije. Jasno je da bi nešto trebalo učiniti.

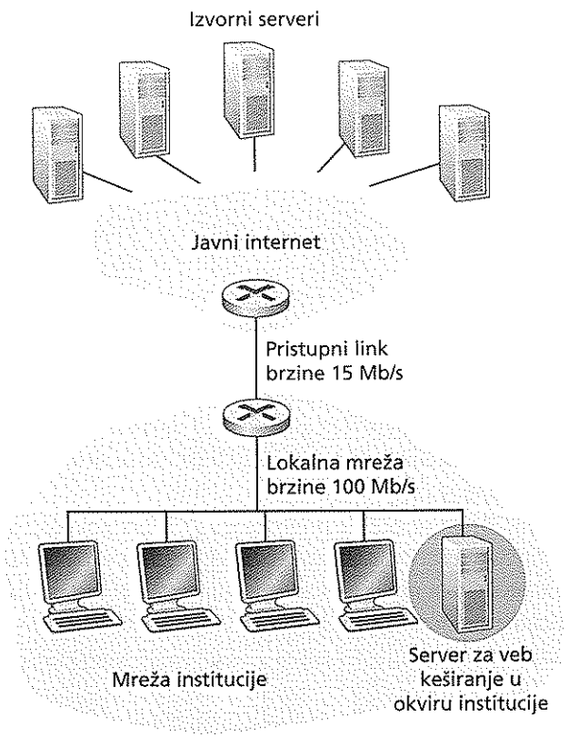
Jedno od mogućih rešenja bilo bi da se brzina pristupa sa 15 Mb/s poveća na, recimo, 100 Mb/s. Ovim će se intenzitet saobraćaja na pristupnom linku smanjiti na 0,15, što znači da će kašnjenja između dva rutera postati zanemarljiva. U ovom slučaju ukupno vreme odziva iznosiće oko dve sekunde, koliko je internet kašnjenje. Međutim, ovo rešenje podrazumeva nadgradnju pristupnog linka sa 15 Mb/s na 100 Mb/s, što može biti i veoma skupo.

Šta bi se dogodilo kada bismo se, umesto za nadgradnju pristupnog linka, opredelili za instaliranje servera za veb keširanje u mreži ove institucije? Ovo rešenje prikazano je na slici 2.13. Stopa uspešno dobijenih podataka iz keš memorije – broj zahteva koje zadovolji server za veb keširanje – u praksi se obično kreće između 0,2 i 0,7. Za potrebe primera pretpostavićemo da keširanje za ovu instituciju pribavlja podatke u udelu od 0,4. Pošto su klijenti i server za veb keširanje povezani istom lokalnom mrežom velike brzine, to znači da će veb keširanje 40 procenata zahteva zadovoljiti praktično trenutno, sa vremenom odziva od, recimo, 10 milisekundi. Preostalih 60 procenata zahteva, ipak, moraju da zadovolje izvorni serveri. Međutim, kako kroz pristupni link sada prolazi samo 60 procenata zahtevanih objekata, intenzitet saobraćaja se sa 1,0 smanjio na 0,6. U praksi, intenzitet saobraćaja manji od 0,8 na linku brzine 15 Mb/s znači manja kašnjenja od, recimo, nekoliko desetina milisekundi. Ovo kašnjenje je zanemarljivo ukoliko se uporedi sa internet kašnjenjem od dve sekunde. Imajući u vidu sve navedene pretpostavke, prosečna vrednost kašnjenja iznosi:

$$0{,}4 \cdot (0{,}01 \text{ sekundi}) + 0{,}6 \cdot (2{,}01 \text{ sekundi})$$

što je tek nešto malo više od 1,2 sekunde. Prema tome, ovo drugo rešenje donosi kraće vreme odziva od prvog, a ne zahteva nadgradnju linka ka internetu. Naravno, institucija mora da kupi i instalira server za veb keširanje, ali to je jeftinije – mnogi

serveri za veb keširanje koriste softver koji se nalazi u javnom domenu i izvršava na jeftinim personalnim računarima.



Slika 2.13 ◆ Dodavanje servera za veb keširanje u mrežu institucije

Korišćenjem **mreže za distribuciju sadržaja tj. CDN mreže** (engl. *Content Distribution Networks – CDN*) veb keširanje igra sve veću ulogu na internetu. CDN kompanija instalira brojne servere za keširanje geografski raspoređene na internetu, i time lokalizuje veći deo saobraćaja. Postoje deljene CDN mreže (kao što su *Akamai* i *Limelight*) i namenske CDN mreže (kao što su *Google* i *Microsoft*). CDN mreže ćemo detaljnije obraditi u poglavlju 7.

2.2.6 Uslovni metod GET

Iako keširanjem može da se primetno skrati vreme odziva, ono uvodi i jedan novi problem – kopija objekta na serveru za veb keširanje može da bude zastarela. Drugim rečima, moguće je da objekat koji se nalazi na veb serveru bude izmenjen nakon što je njegova kopija preneta serveru za veb keširanje. Srećom, HTTP ima mehanizam koji serveru za veb keširanje omogućava da proveri da li su objekti koji se nalaze

na njemu ažurirani. Ovaj mehanizam naziva se **uslovni metod GET**. HTTP poruka zahteva naziva se uslovna GET poruka ukoliko: (1) poruka zahteva koristi metod `GET` i (2) poruka zahteva sadrži red zaglavlja `If-Modified-Since:`.

Da bismo vam prikazali kako uslovni metod `GET` radi, poslužićemo se jednim primerom. Prvo, u ime veb pretraživača koji šalje zahtev, proksi server šalje poruku zahteva odgovarajućem veb serveru:

```
GET /fruit/kiwi.gif HTTP/1.1
Host: www.exotiquecuisine.com
```

Drugo, veb server šalje poruku odgovora i traženi objekat serveru za veb keširanje:

```
HTTP/1.1 200 OK
Date: Thu, 8 Oct 2007 15:39:29
Server: Apache/1.3.0 (Unix)
Last-Modified: Wed, 7 Sep 2011 09:23:24
Content-Type: image/gif

(podaci podaci podaci podaci podaci ...)
```

Server za veb keširanje prosleđuje odgovarajući objekat pretraživaču koji ga je tražio, ali ga i smešta u svoju memoriju. Što je još važnije, server za veb keširanje zajedno sa ovim objektom čuva i datum njegove poslednje izmene. Treće, nedelju dana kasnije neki drugi pretraživač zatraži isti objekat preko ovog servera za veb keširanje, a objekat je još u serveru za veb keširanje. Pošto je taj objekat možda izmenjen u toku tih nedelju dana, server za veb keširanje proverava njegovu aktuelnost, koristeći uslovni metod GET. Tačnije, server za veb keširanje šalje sledeću poruku:

```
GET /fruit/kiwi.gif HTTP/1.1
Host: www.exotiquecuisine.com
If-modified-since: Wed, 7 Sep 2011 09:23:24
```

Obratite pažnju na to da je vrednost reda zaglavlja `If-modified-since:` istovetna kao vrednost reda zaglavlja `Last-Modified:` iz poruke koju je server poslao pre nedelju dana. Ovakvom uslovnom metodom `GET`serveru se kaže da traženi objekat pošalje, samo ukoliko je u međuvremenu izmenjen. Pretpostavimo da objekat nije menjan od 7. septembra 2011. godine u 9 sati 23 minuta i 24 sekunde. U tom slučaju, veb server odgovara sledećom porukom:

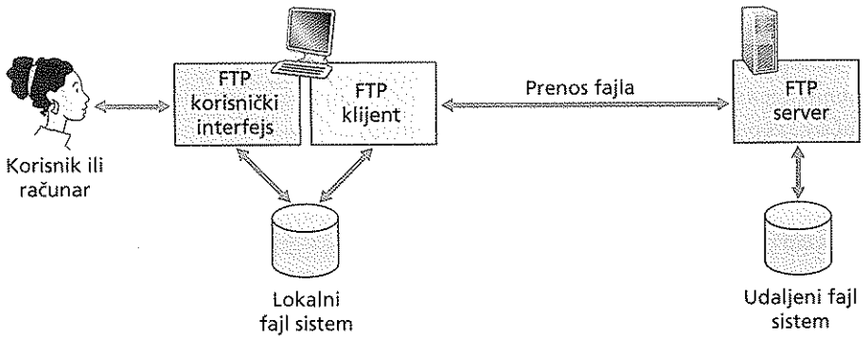
```
HTTP/1.1 304 Not Modified
Date: Sat, 15Oct 2011 15:39:29
Server: Apache/1.3.0 (Unix)

(prazno telo poruke)
```

Vidimo da odgovarajući na uslovni metod GET, veb server i dalje šalje poruku odgovora, ali da u toj poruci odgovora nema traženog objekta. Stavljanjem traženog objekta bespotrebno bi se trošio propusni opseg, a odziv bi bio sporiji, naročito ukoliko je reč o većem objektu. Obratite pažnju na to da se u poslednjoj poruci nalazi i statusni red `304 Not Modified`, kojim se serveru za veb keširanje saopštava da može da prosledi kopiju tog objekta koja se već nalazi u njegovoj memoriji (memoriji proksi servera) pretraživaču koji je tražio objekat.

Ovim završavamo našu priču o protokolu HTTP, prvom internet protokolu (protokolu aplikativnog sloja) koji smo podrobnije proučili. Videli smo format HTTP poruka i postupaka koje preuzimaju veb klijenti i serveri prilikom slanja i prijema ovih poruka. Takođe, delimično smo obradili infrastrukturu veb aplikacija, između ostalih servere za veb keširanje, kolačiće i baze podataka koje se nalaze na veb serverima, pri čemu su svi oni na neki način povezani sa protokolom HTTP.

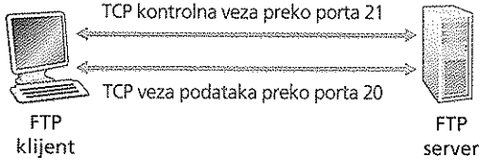
2.3 Prenos datoteka: protokol FTP

Uobičajena sesija za prenos datoteka korišćenjem protokola FTP obavlja se tako što korisnik sedi ispred jednog (lokalnog) računara i želi da prenese datoteke na udaljeni računar, ili da sa njega prenese datoteke na svoj računar. Da bi pristupio udaljenom računaru, korisnik mora da se identifikuje i unese svoju lozinku. Pošto obezbedi ove informacije, korisnik može da počne sa prenosom datoteka sa lokalnog sistema datoteka na udaljeni sistem datoteka i obratno. Kao što je prikazano na slici 2.14, korisnik upotrebljava protokol FTP, koristeći odgovarajući program za prenos datoteka. Korisnik najpre upisuje ime udaljenog računara, nakon čega klijentski proces protokola FTP na lokalnom računaru uspostavlja TCP vezu sa serverskim procesom protokola FTP na udaljenom računaru. Posle toga, korisnik upisuje svoje korisničko ime i lozinku, koji se šalju TCP vezom kao deo FTP komandi. Kada server proveri korisnika i dozvoli mu pristup, taj korisnik može da počne sa kopiranjem jedne ili više datoteka sa lokalnog sistema datoeka na udaljeni sistem datoteka (ili obratno).



Slika 2.14 ◆ Prenos datoteka između lokalnog i udaljenog sistema datoteka protokolom FTP

Protokoli HTTP i FTP su protokoli za prenos datoteka i imaju mnogo zajedničkih osobina; na primer, oba protokola rade preko protokola TCP. Ipak, između ova dva protokola aplikativnog sloja postoje i neke bitne razlike. Najočiglednija razlika je to što protokol FTP za prenos datoteka koristi dve paralelne TCP veze – **kontrolnu vezu** i **vezu podataka**. Kontrolna veza se koristi za razmenu kontrolnih informacija između dva računara – informacija kao što su identitet korisnika i njegova lozinka, komande za izmenu udaljenog direktorijuma i komande za „stavljanje" i „uzimanje" (eng. put i get) datoteka. Veza podataka se u suštini koristi za prenos datoteka. Pošto protokol FTP koristi zasebnu kontrolnu vezu, za njega se kaže da svoje kontrolne informacije šalje **izvan opsega**. Sećate se da protokol HTTP redove zaglavlja zahteva i odgovora šalje istom TCP vezom kojom se prenosi i sama datoteka. Zato se za protokol HTTP kaže da svoje kontrolne informacije šalje **u opsegu**. U sledećem odeljku videćemo da protokol SMTP, osnovni protokol za elektronsku poštu, svoje kontrolne informacije takođe šalje u opsegu. Na slici 2.15 su prikazane kontrolna veza i veza podataka protokola FTP.



Slika 2.15 ◆ Kontrolna veza i veza podataka

Kada korisnik započne, sesija protokola FTP sa udaljenog računara, klijentska strana protokola FTP (korisnik) prvo uspostavlja kontrolnu TCP vezu sa serverskom stranom (udaljenim računarom) preko serverovog porta broj 21. Klijentska strana protokola FTP ovom kontrolnom vezom šalje identifikaciju i lozinku korisnika, kao i komande za menjanje udaljenog direktorijuma. Kada serverska strana primi komande za prenos datoteka (sa udaljenog računara, ili na njega), ona uspostavlja TCP vezu podataka sa klijentskom stranom. Protokol FTP vezom podataka šalje samo jednu datoteku i zatim je prekida. Ukoliko, tokom iste sesije, korisnik želi da prebaci još neku datoteku, protokol FTP uspostavlja još jednu vezu podataka. Dakle, protokol FTP radi tako što kontrolnu vezu ostavlja otvorenom, sve vreme dok traje sesija korisnika, a za svaku novu datoteku unutar sesije uspostavlja se nova veza podataka (to jest, veze podataka nisu postojane).

Za sve vreme rada FTP server mora da vodi računa o **stanju** korisnika. Tačnije rečeno, server mora određenu kontrolnu vezu da pridruži tačno određenom korisničkom nalogu i mora da prati tekući direktorijum korisnika, dok se korisnik kreće kroz stablo direktorijuma na udaljenom računaru. Praćenje informacija o stanju sesije za sve trenutno povezane korisnike značajno ograničava njihov ukupan broj, koji protokol FTP može da podržava istovremeno. Sećate se da protokol HTTP, s druge strane, radi bez nadgledanja stanja – on ne mora da vodi računa o stanju korisnika.

2.3.1 FTP komande i odgovori

Ovaj odeljak završićemo kraćim opisom FTP komandi i odgovora koji se najčešće koriste. Komande, od klijenta do servera, i odgovori, od servera do klijenta, šalju se kontrolnom vezom u sedmobitnom ASCII formatu. To znači da ove komande, poput HTTP komandi, ljudi mogu da pročitaju. Da bi se susedne komande razdvojile, iza svake komande nalaze se: znak za povratak na početak reda i znak za novi red. Svaka komanda sastoji se od četiri ASCII znaka, ispisana velikim slovima, od kojih su neke sa opcionim argumentima. Evo nekih komandi koje se najčešće upotrebljavaju:

- `USER ime korisnika`: koristi se za slanje identifikacije korisnika do servera.
- `PASS lozinka`: koristi se za slanje lozinke korisnika serveru,
- LIST: koristi se da bi se od servera zatražilo da pošalje spisak svih datoteka iz tekućeg direktorijuma na udaljenom računaru; ovaj spisak šalje se (novom i nepostojanom) vezom podataka, a ne TCP kontrolnom vezom.
- `RETR naziv datoteke`: koristi se za preuzimanje (to jest, dobijanje) datoteke iz tekućeg direktorijuma na udaljenom računaru; ova komanda dovodi do uspostavljanja veze podataka sa udaljenog računara i do slanja tražene datoteke tom vezom.
- `STOR naziv datoteke`: koristi se za snimanje (to jest, stavljanje) datoteke u tekući direktorijum na udaljenom računaru.

Obično, između komande koju korisnik zadaje i FTP komande koja se šalje kontrolnom vezom, postoji preslikavanje jedan na jedan. Na sve komande sledi odgovor koji server pošalje klijentu. Ovi odgovori su trocifreni brojevi iza kojih se po potrebi nalazi poruka. Po tome podsećaju na statusne kodove i fraze u statusnom redu HTTP poruke odgovora. Evo nekih uobičajenih odgovora zajedno sa mogućim porukama:

- `331UsernameOK,passwordrequired`
- `125Dataconnectionalreadyopen;transferstarting`
- `425Can'topendataconnection`
- `452Errorwritingfile`

Čitaoce koji žele da upoznaju i druge FTP komande i odgovore upućujemo da pročitaju dokument RFC 959.

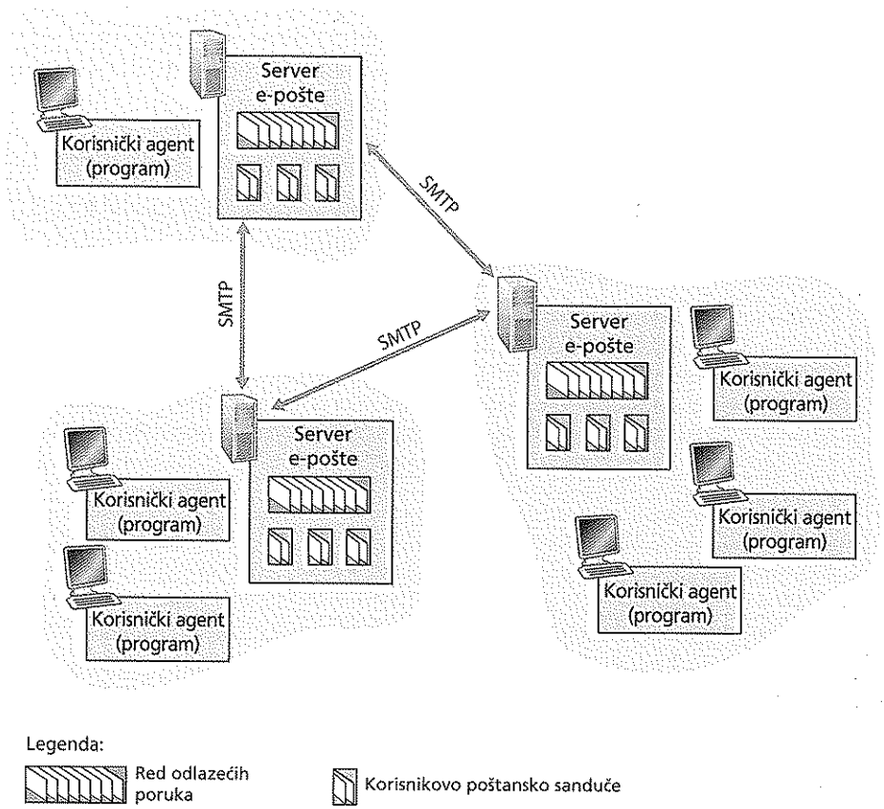
2.4 Elektronska pošta na internetu

Elektronska pošta postoji od samih početaka interneta i u tim najranijim danima interneta bila je najpopularnija aplikacija [Segaller 1998]. Vremenom, e-pošta se

sve više razvijala i postajala sve moćnija, a još uvek je jedna od najvažnijih i naviše korišćenih aplikacija.

Poput obične poštanske službe, e-pošta je asinhroni komunikacioni medijum – ljudi šalju i čitaju poruke onda kada im to odgovara i ne moraju da usklađuju vreme sa drugim ljudima. Ali, za razliku od obične pošte, e-pošta je brza, jednostavno se distribuira i jeftina. Savremena e-pošta ima mnogo korisnih osobina, uključujući poruke sa prilozima, hiperveze, tekst u HTML formatu, ili ugrađene slike.

U ovom odeljku istražićemo protokole aplikativnog sloja koji predstavljaju srce e-pošte na internetu. Ipak, pre nego što se upustimo u podrobnije razmatranje ovih protokola, razmotrimo najopštiji prikaz sistema za e-poštu na internetu i njegove ključne delove.



Slika 2.16 ◆ Najopštiji prikaz sistema za e-poštu na internetu

Slika 2.16 predstavlja najopštiji prikaz sistema za e-poštu na internetu. Na ovom crtežu vidimo da ovaj sistem ima tri osnovna dela – **korisničke agente**, **servere e-pošte** i **protokol SMTP** (Simple Mail Transfer Protocol – jednostavan protokol za prenos

pošte). Sada ćemo svaki od ovih delova opisati na primeru pošiljaoca, Alise, koja šalje e-poštu primaocu, Bobu. Korisnički agenti omogućavaju korisnicima čitanje poruka, odgovaranje na njih, kao i njihovo prosleđivanje, snimanje i sastavljanje. *Microsoft Outlook* i *Apple Mail* su primeri za korisničke agente e-pošte. Kada Alisa završi pisanje svoje poruke, njen korisnički agent šalje ovu poruku njenom serveru za e-poštu, gde se poruka smešta u red odlazećih poruka servera e-pošte. Kada Bob bude želeo da pročita ovu poruku, njegov korisnički agent preuzima tu poruku iz njegovog poštanskog sandučeta na njegovom serveru za elektronsku poštu.

Serveri e-pošte čine osnovu insfrastrukture e-pošte. Svaki primalac, kao Bob, ima **poštansko sanduče** koje se nalazi u jednom od servera e-pošte. Bobovo poštansko sanduče upravlja i vodi računa o porukama koje su mu poslate. Uobičajena poruka svoje putovanje započinje u korisničkom agentu pošiljaoca, putuje do pošiljaočevog servera za e-poštu, pa zatim do primaočevog servera za e-poštu, gde se smešta u poštansko sanduče primaoca.

ISTORIJSKA ČITANKA

VEB E-POŠTA

Decembra 1995. godine, samo nekoliko godina od „otkića" veba, Sabir Bhatia i Džek Smit posetili su internet investitora, Drapera Fišera Jurvetsona i predložili mu razvijanje besplatnog sistema za e-poštu koji bi se zasnivao na vebu. Ideja je bila da se svakome, ko to želi, dodeli besplatan nalog za e-poštu, a da se tim nalozima omogući pristup preko veba. U zamenu za 15 procenata udela u *kompaniji*, Draper Fišer Jurvetson je finansirao osnivanje kompanije pod nazivom *Hotmail*. Sa troje stalno zaposlenih i 14 povremeno zaposlenih koji su radili za udeo u deonicama kompanije, jula 1996. godine uspeli su da razviju i pokrenu ovu uslugu. U toku prvog meseca od pokretanja imali su 100 000 pretplatnika. U decembru 1997. godine, manje od 18 meseci pošto su pokrenuli ovu uslugu, *Hotmail* je imao preko 12 miliona korisnika, a onda ga je kupio *Microsoft*, zvanično za 400 miliona dolara. Uspeh *Hotmail*-a obično se pripisuje njegovoj „prednosti prvog pokretanja" i intrističnom „viralnom marketingu" e-pošte. Možda će neki od studenata koji čitaju ovu knjigu biti među novim preduzetnicima koji će osmisliti i razviti nedostižnu internet uslugu svojstvenu viralnom marketingu.

Veb e-pošta nastavlja da se razvija i postaje svake godine sofisticiranija i još moćnija. Jedna od najpopularnijih usluga danas je *Google*-ov *gmail*, koji nudi: gigabajte slobodnog skladišta, napredno filtriranje nepoželjne pošte i otkrivanje virusa, šifrovavanje e-pošte pomoću SSL protokola (sloja bezbednih priključaka, engl. *Secure Socket Layer*), preuzimanje pošte iz usluga e-pošte treće strane, kao i interfejse orijentisane na pretraživanje. Asinhrono slanje poruka unutar društvenih mreža, kao što je *Facebook*, postalo je takođe popularno poslednjih godina.

Kada Bob poželi da pristupi porukama koje se nalaze u njegovom poštanskom sandučetu, server za e-poštu na kome se nalazi njegovo poštansko sanduče proverava Bobova ovlašćenja (koristeći korisničko ime i lozinku). Alisin server za e-poštu takođe mora da se izbori i sa mogućim problemima na Bobovom serveru za e-poštu. Ukoliko njen server ne može da isporuči poštu Bobovom serveru, on će poruku zadržati u **redu poruka** i pokušati da je isporuči kasnije. Ovi pokušaji se ponavljaju na svakih tridesetak minuta; ukoliko ni nakon nekoliko dana ne uspe da isporuči poruku, server je uklanja i o tome e-porukom obaveštava pošiljaoca (Alisu).

Protokol SMTP predstavlja osnovni protokol aplikativnog sloja za elektronsku poštu na internetu. Prilikom prenosa poruka od pošiljaočevog do primaočevog servera za e-poštu ovaj protokol koristi uslugu pouzdanog prenosa podataka protokola TCP. Poput većine protokola aplikativnog sloja i protokol SMTP ima dve strane – klijentsku, koja se izvršava na serveru za e-poštu pošiljaoca, i serversku, koja se izvršava na serveru za e-poštu primaoca. Obe strane ovog protokola izvršavaju se na svim serverima za e-poštu. Kada server za elektronsku poštu šalje poruke drugim serverima za e-poštu, ponaša se kao SMTP klijent. Kada server za e-poštu prima poruke od drugih servera za e-poštu, ponaša se kao SMTP server.

2.4.1 Protokol SMTP

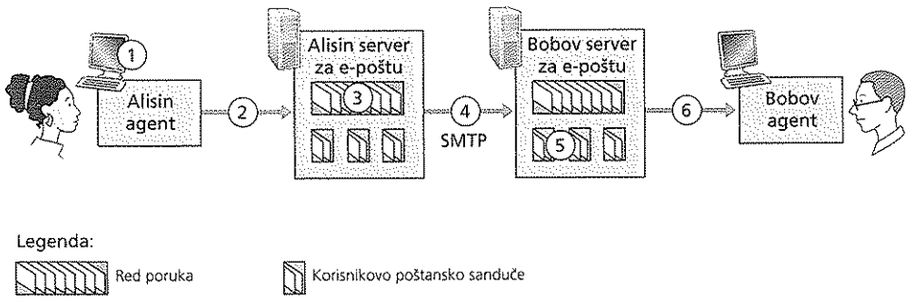
Protokol SMTP, definisan dokumentom RFC 2821, predstavlja srce elektronske pošte na internetu. Kao što smo već rekli, SMTP prenosi poruke od pošiljaočevog do primaočevog servera za e-poštu. Protokol SMTP je znatno stariji od protokola HTTP. (Prvobitni dokument RFC za protokol SMTP pojavio se 1982. godine, ali se sâm SMTP pojavio znatno ranije.) Iako se protokol SMTP odlikuje mnogim izuzetnim osobinama, o čemu svedoči i to da je na internetu neizbežan, ipak je tu reč o pomalo zastareloj tehnologiji koja ima i neke arhaične karakteristike. Na primer, ovaj protokol ograničava telo (ne samo zaglavlje) svih poruka na sedmobitni ASCII kôd. Ovo ograničenje je imalo smisla početkom osamdesetih, kada je prenosni kapacitet bio skroman i kada niko nije slao kao dodatak e-poruke velike slike, ili audio i video datoteke. Ali danas, u multimedijalnoj eri, ograničenje na sedmobitni ASCII kôd prilično smeta – binarni multimedijalni podaci moraju da se koduju u ASCII kôd, kako bi mogli da se prenose protokolom SMTP; a nakon SMTP transporta ASCII poruka mora da se dekoduje i vrati u prvobitni binarni oblik. Podsećamo vas da smo u odeljku 2.2 rekli da protokol HTTP ne zahteva ovu vrstu kodovanja pre prenosa.

Da bismo prikazali način rada protokola SMTP, poslužićemo se jednim uobičajenim slučajem. Pretpostavimo da Alisa želi da Bobu pošalje jednostavnu ASCII poruku.

1. Alisa pokreće svoj korisnički agent za e-poštu, upisuje Bobovu e-adresu (na primer, `bob@nekaskola.edu`), sastavlja poruku i zatim nalaže korisničkom agentu da je pošalje.

2. Alisin korisnički agent šalje poruku njenom serveru za e-poštu, na kome se ona smešta u red poruka.
3. Klijentska strana protokola SMTP, koja se izvršava na Alisinom serveru za e-poštu, uočava ovu poruku u redu poruka i uspostavlja TCP vezu sa serverom protokola SMTP, koji se izvršava na Bobovom serveru za e-poštu.
4. Posle uobičajenog početnog uspostavljanja veze dve strane protokola SMTP, klijentska strana protokola SMTP šalje Alisinu poruku TCP vezom.
5. Na Bobovom serveru za e-poštu, serverska strana protokola SMTP prima poruku. Bobov server za e-poštu smešta tu poruku u Bobovo poštansko sanduče.
6. Bob pokreće svoj korisnički agent, onda kada on želi da pročita svoju poštu.

Ovaj postupak prikazan je na slici 2.17.



Slika 2.17 ◆ Alisa šalje poruku Bobu

Veoma je važno da uočite to da protokol SMTP za slanje poruka ne koristi posredničke servere za e-poštu, čak i kada se dva servera za e-poštu nalaze na suprotnim krajevima sveta. Ukoliko se Alisin server za e-poštu nalazi u Hong Kongu, a Bobov u Sent Luisu, između njih se uspostavlja direktnaTCP veza. To znači da, ukoliko je Bobov server privremeno u kvaru, poruka ostaje na Alisinom serveru za e-poštu i čeka novu priliku za slanje – poruka se ne ostavlja na nekom posredničkom serveru za e-poštu.

Pogledajmo sada izbliza kako protokol SMTP prenosi poruku sa servera za e-poštu pošiljaoca na server za e-poštu primaoca. Videćemo da protokol SMTP ima mnogo sličnosti sa pravilima koja se koriste u uobičajenom razgovoru između ljudi. Prvo, klijentska strana protokola SMTP (koja se izvršava na serveru za e-poštu hosta) mora da uspostavi TCP vezu preko porta25, serverske strane protokola SMTP (koja se izvršava na serveru za e-poštu). Ukoliko je server u kvaru, klijent kasnije pokušava ponovo. Pošto uspostave ovu vezu, klijentska i serverska strana protokola međusobno se usklađuju (eng. handshaking) na aplikativnom sloju – baš kao što se ljudi predstavljaju pre nego što počnu razgovor, tako se i SMTP klijent i serveri

predstavljaju pre nego što počnu da razmenjuju informacije. Tokom ove faze usklađivanja, SMTP klijent navodi adresu za e-poštu pošiljaoca (osobe koja je napravila poruku) i adresu za e-poštu primaoca. Pošto se SMTP klijent i server predstave jedan drugom, klijent šalje poruku. Protokol SMTP se za prenos poruke bez greške oslanja na uslugu pouzdanog prenosa podataka koju nudi protokol TCP. Klijent zatim ponavlja ovaj postupak preko iste TCP veze, ukoliko ima još poruka za slanje na taj server; u suprotnom, nalaže protokolu TCP da prekine vezu.

Pogledajmo sada kako izgleda primer transkripta razmene poruka između SMTP klijenta (C) i SMTP servera (S). Ime računara klijenta je `crepes.fr`, a ime računara servera je `hamburger.edu`. Redovi ASCII teksta ispred kojih stoji `C:` su oni koje klijent šalje u svoj TCP soket, dok su redovi ASCII teksta ispred kojih stoji `S:` oni koje server šalje u svoj TCP soket. Transkript koji sledi započinje neposredno nakon uspostavljanja TCP veze.

```
S:220 hamburger.edu
C:HELOcrepes.fr
S:250Hellocrepes.fr,pleasedtomeetyouC:MAILFROM: <alice@crepes.fr>
S:250alice@crepes.fr...SenderokC:RCPTTO: <bob@hamburger.edu>
S:250bob@hamburger.edu...RecipientokC:DATA
S:354Entermail,endwith".”onalinebyitselfC:Doyoulikeketchup?
C:Howaboutpickles?C:.
S:250MessageacceptedfordeliveryC:QUIT
S:221 hamburger.edu closing connection
```

U ovom primeru, klijent sa servera za e-poštu `crepes.fr` šalje poruku („`Do you like ketchup? How about pickles?`”) do servera za e-poštu `hamburger.edu`. Kao deo razgovora klijent je izdao pet komandi: `HELO` (skraćeno od `HELLO`), `MAIL FROM`, `RCPT TO`, `DATA` i `QUIT`. Ove komande su jasne same po sebi. Osim toga, klijent je poslao i red u kome se nalazi samo jedna tačka, kojom se naznačava kraj poruke upućene serveru. (U ASCII žargonu svaka poruka se završava sa `CRLF.CRLF`, gde se `CR` odnosi na povratak na početak reda, a `LF` na prelazak u novi red.) Server na svaku od ovih komandi daje odgovore koji imaju odgovarajući kôd i neko (opciono) objašnjenje na engleskom jeziku. Na ovom mestu reći ćemo i to da protokol SMTP koristi postojane veze: ukoliko bi server za e-poštu pošiljaoca trebalo da pošalje serveru za e-pošte primaoca nekoliko poruka, sve ih šalje istom TCP vezom. Svaku poruku klijent započinje novom komandom `MAIL FROM:crepes.fr`, njen kraj označava izdvojenom tačkom, a komandu `QUIT` izdaje tek nakon što pošalje sve poruke.

Preporučuje se da za neposredan pristup nekom SMTP serveru koristite program *Telnet*. Da biste to učinili, potrebno je da izdate sledeću komandu:

```
telnet server Name 25;
```

gde `server Name` označava ime lokalnog servera za e-poštu. Ovim ste uspostavili TCP vezu između vašeg lokalnog računara i servera za e-poštu. Odmah pošto upišete ovaj red trebalo bi da od servera primite odgovor `220`. Posle toga izdajete SMTP komande: `HELO`, `MAIL FROM`, `RCPT TO`, `DATA`, `CRLF`. Komande `CRLF` i `QUIT` izdajete onda kada je to potrebno. Na kraju ovog poglavlja obavezno provežbajte programerski zadatak 3, u kome bi trebalo da napravite jednostavan korisnički agent koji primenjuje klijentsku stranu protokola SMTP. Pomoću njega moći ćete da šaljete poruke bilo kom primaocu putem lokalnog servera za e-poštu.

2.4.2 Poređenje sa protokolom HTTP

Ukratko ćemo uporediti protokole SMTP i HTTP. Oba protokola koriste se za prenos datoteka sa jednog računara na drugi: HTTP prenosi datoteke (nazivaju se i objektima) od veb servera do veb klijenta (najčešće pretraživača); SMTP prenosi datoteku (elektronsku poruku) od jednog servera za e-poštu do drugog. Prilikom prenosa datoteka postojani protokol HTTP i SMTP koriste postojane veze. Prema tome, moglo bi se reći da ova dva protokola imaju neke zajedničke karakteristike. Međutim, između njih postoje i veoma značajne razlike. Prvo, protokol HTTP je u najvećoj meri **prijemni** (engl. *pull*) **protokol** – neko postavlja određene informacije na veb serveru, a korisnici pomoću protokola HTTP primaju te informacije sa servera, onda kada oni to žele. Tačnije, TCP vezu pokreće onaj računar koji želi da prima podatke. S druge strane, protokol SMTP je prvenstveno **predajni** (engl. *push*) **protokol** – server za e-poštu pošiljaoca predaje datoteku serveru za e-poštu primaoca. U ovom slučaju TCP vezu uspostavlja onaj računar koji želi da pošalje određenu datoteku.

Druga razlika, koju smo već pomenuli, jeste to što protokol SMTP zahteva da svaka poruka, uključujući tu i telo svake poruke, bude u sedmobitnom ASCII formatu. Ako poruka sadrži karaktere koji nisu sedmobitni ASCII kôd (na primer, francuska slova sa akcentima), ili sadrže binarne podatke (recimo, datoteke slike), onda ona mora da se koduje u sedmobitni ASCII kôd. Protokol HTTP ne nameće ovo ograničenje.

Treća bitna razlika odnosi se na način postupanja sa dokumentima koji sadrže tekst i slike (i možda neke druge vrste multimedijalnih podataka). Kao što ste videli u odeljku 2.2, protokol HTTP enkapsulira svaki objekat u sopstvenu poruku odgovora. E-pošta na internetu, sve objekte poruke smešta u jednu poruku.

2.4.3 Formati elektronske pošte

Kada Alisa piše Bobu obično pismo, ona u zaglavlju na njegovom vrhu može da upiše razne vrste dodatnih informacija, kao što su Bobova adresa, njena povratna adresa i datum. Slično tome, prilikom slanja e-pošte od jedne osobe drugoj, zaglavlje sa dodatnim informacijama prethodi telu same poruke. Ove dodatne informacije nalaze se u nizu redova zaglavlja koja su definisana dokumentom RFC 5322. Redovi zaglavlja su odvojeni od tela poruke praznim redom (odnosno znakom `CRLF`). Dokument RFC 5322 utvrđuje tačan format redova zaglavlja pošte, kao i njihove semantičke interpretacije. Poput protokola HTTP, svaki red zaglavlja sadrži neki čitljiv tekst koji čini ključna reč praćena znakom dve tačke i vrednošću. Neke ključne reči su obavezne, dok su druge neobavezne. Svako zaglavlje mora da ima redove zaglavlja `From:` i `To:`, može da sadrži red zaglavlja `Subject:`, kao i ostale redove zaglavlja koji nisu obavezni. Ovde želimo da naglasimo da se ovi redovi zaglavlja *razlikuju* od SMTP komandi o kojima smo govorili u odeljku 2.4.1 (i pored toga što se i kod jednih i kod drugih javljaju reči „*from*" i „*to*"). Komande o kojima smo govorili u tom odeljku bile su deo usklađivanja protokola SMTP; redovi zaglavlja o kojima sada pričamo čine delove same poruke e-pošte.

Uobičajeno zaglavlje poruke izgleda ovako:

```
From:alice@crepes.fr
To:bob@hamburger.edu
Subject:Searchingforthemeaningoflife.
```

Nakon zaglavlja poruke sledi prazan red, pa zatim telo poruke (u ASCII kodu). Preporučujemo vam da pomoću programa *Telnet* pošaljete serveru za e-poštu poruku koja sadrži neke redove zaglavlja, uključujući i red zaglavlja `Subject:`. Da biste to uradili, izdajte komandu `telnet server Name 25`, kao što smo objasnili u poglavlju 2.4.1.

2.4.4 Protokoli za pristupanje pošti

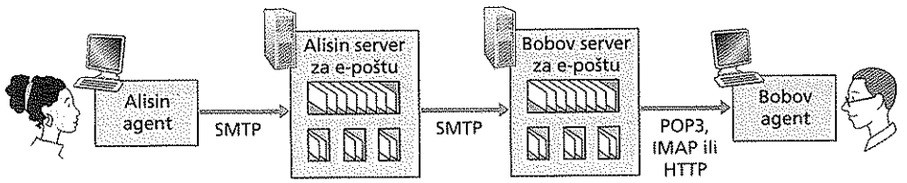
Kada protokol SMTP isporuči neku poruku sa Alisinog servera za e-poštu Bobovom serveru za e-poštu, ta poruka se smešta u Bobovo poštansko sanduče. Kroz čitavo izlaganje prećutno smo podrazumevali da Bob čita svoje poruke tako što se prijavljuje na određeni računar – server, a zatim na tom računaru pokreće čitač e-pošte. Sve do početka devedesetih ovo je bio i uobičajeni način da se to uradi. Međutim, danas se e-porukama pristupa korišćenjem klijent-server arhitekture – tipični korisnik čita svoju e-poštu pomoću klijentskog programa koji se izvršava na njegovom krajnjem sistemu, što može da bude njegov računar u kancelariji, laptop, ili pametni telefon. Izvršavanje klijenata za e-poštu na lokalnim računarima donelo je krajnjim korisnicima čitav niz novih mogućnosti, uključujući i pregledanje multimedijalnih poruka i priloga.

Pošto Bob (primalac) izvršava svoj korisnički agent na svom lokalnom računaru, samo po sebi se nameće i to da se i server za e-poštu nađe na Bobovom lokalnom

računaru. Ovakvim rešenjem, Alisin server za e-poštu mogao bi neposredno da razmenjuje podatke sa Bobovim računarom. Međutim, ovde postoji i jedan problem. Ako se sećate, rekli smo da server za e-poštu upravlja poštanskim sandučićima i da se na njemu izvršavaju klijentska i serverska strana protokola SMTP. Kada bi se Bobov server za e-poštu nalazio na njegovom lokalnom računaru, taj računar bi morao da bude neprekidno uključen i povezan sa internetom, jer bi jedino tako mogao da primi poštu koja može da pristigne bilo kad. Ovo nije praktično za većinu korisnika interneta. Umesto toga, prosečan korisnik pokreće korisnički agent na lokalnom računaru, ali pristupa svom poštanskom sandučetu koje se nalazi na stalno uključenom zajedničkom serveruza e-poštu. Ovakve servere za e-poštu zajednički koristi veći broj korisnika, a o njima obično vodi računa njihov posrednik za internet usluge (recimo, univerzitet ili neka kompanija).

Pogledajmo sada kako izgleda putanja kojom e-poruka putuje od Alise do Boba. Znamo da se u nekom delu ove putanje e-poruka mora odložiti naBobov server za e-poštu. Ovo se postiže veoma jednostavno – tako što Alisin korisnički agent neposredno šalje poruku Bobovom serveru za e-poštu. To može da obavi protokol SMTP koji je upravo i napravljen za predaju e-pošte između računara. Međutim, korisnički agent pošiljaoca najčešće ne ostvaruje neposrednu vezu sa serverom za e-poštu primaoca. Umesto toga, kao šo je prikazano na slici 2.18, Alisin korisnički agent koristi SMTP za prebacivanje e-poruke na njen server za e-poštu, a zatim Alisin server za e-poštu koristi SMTP (kao SMTP klijent) za prenošenje te e-poruke do Bobovog servera za e-poštu. Zbog čega se to radi u dva poteza? Prvenstveno zato što Alisin korisnički agent ne bi mogao da isporuči e-poruku trenutno nedostupnom odredišnom serveru za e-poštu bez posredovanja njenog servera za e-poštu. Time što Alisa prvo odloži e-poruku na sopstveni server za e-poštu, ona omogućava da taj server svakih 30 minuta pokušava da pošalje poruku, sve dok Bobov server za e-poštu ne bude ponovo dostupan. (U slučaju kvara njenog servera za e-poštu, ona uvek može da se žali administratoru sistema.) RFC dokumentima o protokolu SMTP uređen je način korišćenja SMTP komandi za prosleđivanje poruke između većeg broja SMTP servera.

Ipak, nedostaje nam još jedan deo čitave slagalice. Kako primalac poput Boba, kada pokrene korisnički agent na svom lokalnom računaru, stiže do poruka koje se nalaze na serveru za e-poštu njegovog posrednika za internet usluge? Skrećemo vam pažnju na to da Bobov korisnički agent ne može iskoristi SMTP za dobija-



Slika 2.18 ◆ Protokoli e-pošte i njihovi delovi koji komuniciraju

nje poruka, zato što je dobijanje poruka postupak povlačenja podataka, a protokol SMTP je protokol za predaju podataka. Slagalica je gotova uvođenjem posebnog protokola za pristupanje porukama koji prenosi poruke od Bobovog servera za e-poštu do njegovog lokalnog računara. Trenutno je popularno više ovakvih protokola, uključujući i **POP3 (Post Office Protocol – Version 3)**, **IMAP (Internet Mail Access Protocol)** i HTTP.

Na slici 2.18 možete da vidite protokole koji se koriste za e-poštu na internetu: SMTP se koristi za prenos pošte od servera; za e-poštu pošiljaoca do servera; za e-poštu primaoca, i takođe za prenos pošte od korisničkog agenta pošiljaoca do servera za e-poštu primaoca. Protokoli za pristupanje porukama, kao što je POP3, koriste se za prenos pošte od servera za e-poštu primaoca do njegovog korisničkog agenta.

POP3

Protokol POP3 je izuzetno jednostavan protokol za pristupanje e-pošti. Definisan je u dokumentu [RFC 1939], koji je veoma kratak i razumljiv. Zbog ove jednostavnosti, mogućnosti ovog protokola prilično su ograničene. Protokol POP3 stupa na scenu onog trenutka kada korisnički agent (klijent) otvori TCP vezu sa serverom za e-poštu (server) na portu 110. Pošto se uspostavi TCP veza, POP3 prolazi kroz tri faze: autorizaciju, transakciju i ažuriranje. Tokom prve faze, autorizacije, korisnički agent šalje korisničko ime i lozinku (u čitljivom obliku) kojim se proverava identitet korisnika za rad. Tokom druge faze, transakcije, korisnički agent preuzima poruke, a tokom ove faze može i da obeleži poruke za brisanje, ukloni oznake za brisanje i dobije statističke podatke o e-pošti. Treća faza, ažuriranje, nastupa nakon što klijent izda komandu `quit`, kojom se završava sesija protokola POP3; u ovom trenutku server za e-poštu briše poruke koje su obeležene za brisanje.

U fazi transakcije protokola POP3 korisnički agent izdaje komande na koje server odgovara. Postoje dva moguća odgovora: `+OK` (nekada odmah iza slede podaci koje server šalje klijentu), kojom server odgovara da je sa prethodnom komandom sve u redu i `-ERR`, kojom ukazuje da nešto nije u redu sa prethodnom komandom.

Faza autorizacije ima dve osnovne komande: `user <ime korisnika>` i `pass <lozinka>`. Da biste se upoznali sa ovim komandama, preporučujemo da, koristeći port 110, uspostavite neposrednu Telnet sesiju sa POP3 serverom i da izdate ove komande. Ukoliko pretpostavimo da je ime vašeg servera za e-poštu `mail Server`, videćemo otprilike ovo:

```
telnetmailServer 110
+OKPOP3serverreadyuserbob
+OK
passhungry
+OKusersuccessfullyloggedon
```

Ukoliko neku komandu napišete pogrešno, POP3 server će odgovoriti porukom `-ERR`.

Usredsredimo se sada na fazu transakcije. Korisnički agent koji koristi protokol POP3 može da se podesi (što čini korisnik) tako da „preuzima i briše", ili „preuzima i čuva" poruke. Redosled komandi koje izdaje POP3 korisnički agent zavisi od toga za koji se od ova dva režima korisnik opredelio. U režimu preuzimanja i brisanja korisnički agent izdaje komande `list`, `retr` i `dele`. Na primer, pretpostavimo da neki korisnik u svom poštanskom sandučetu ima dve poruke. U razmeni poruka koja sledi C:(od klijenta) označava korisničkog agenta, a S:(od servera) server za e-poštu. Razmena poruka izgleda otprilike ovako:

```
C:list
S:1498
S:2912S:.
C:retr1
S:(blahblah...
S:.................
S:..........blah)
S:.
C:dele1
C:retr2
S:(blahblah...
S:.................
S:..........blah)
S:.
C:dele2
C:quit
S:+OKPOP3serversigningoff
```

Korisnički agent najpre traži da server za e-poštu navede veličinu svih sačuvanih poruka. Nakon toga, korisnički agent najpre preuzima, a zatim briše poruke sa servera. Obratite pažnju na to da, nakon faze autorizacije, korisnički agent koristi samo četiri komande: `list`, `retr`, `dele` i `quit`. Sintaksa ovih komandi je definisana u dokumentu RFC 1939. Posle obrade komande `quit`, POP3 server prelazi u fazu ažuriranja i iz poštanskog sandučeta uklanja poruke 1 i 2.

Problem u režimu preuzimanja i brisanja je u tome što primalac, Bob, može da koristi više računara, a da pritom želi da sa svih računara koje koristi (u kući, na poslu i sa prenosivog računara) može da pristupa svojim e-porukama. U ovom režimu rada Bobove poruke se nalaze samo na jednom od ta tri računara: tačnije, ako Bob neku poruku prvo pročita na računaru u kancelariji, neće moći da je uveče, kod kuće, ponovo pročita na svom prenosivom računaru. U režimu preuzimanja i čuvanja, korisnički agent, nakon preuzimanja, ostavlja poruke na serveru za e-poštu. U ovom slučaju Bob može ponovo da pročita svoju e-poštu na nekom drugom računaru; može da je pročita na poslu, a kasnije u toku nedelje i kod kuće.

Tokom POP3 sesije između korisničkog agenta i servera za e-poštu, POP3 server održava neke informacije stanja; tačnije rečeno, vodi računa o tome koje je poruke korisnik obeležio za brisanje. Međutim, POP3 server ove informacije ne prenosi iz jedne sesije u drugu, što značajno pojednostavljuje njegovu primenu.

IMAP

Koristeći POP3, Bob, pošto preuzme poruke na svoj lokalni računar, može da napravi facikle za e-poštu i u njih smesti preuzete poruke. Potom može da ih briše, premešta iz jedne fascikle u drugu i pretražuje (prema imenu pošiljaoca ili temi). Ali ova paradigma – tačnije, fascikle i poruke na lokalnom računaru – predstavlja problem za korisnike u pokretu, kojima bi više odgovaralo da se ovakva hijerarhija fascikli napravi i održava na udaljenom serveru, kome bi mogli da pristupaju sa bilo kog računara. Ovo nije moguće postići sa protokolom POP3 koji korisnicima ne omogućava pravljenje fascikli na udaljenim serverima i raspoređivanje poruka u te fascikle.

Da bi se rešio ovaj, ali i drugi problemi, izmišljen je protokol IMAP (Internet Mail Access Protocol) koji je definisan u dokumentu [RFC 3501]. Poput protokola POP3, i IMAP je protokol za pristup e-pošti. Međutim, on ima daleko više mogućnosti od protokola POP3 i znatno je složeniji. (Stoga je njegova realizacija klijentske i serverske strane mnogo složenija.)

IMAP server svaku poruku povezuje sa određenom fasciklom; čim poruka dospe na server, ona se povezuje sa primaočevom fasciklom INBOX. Nakon toga primalac može da je premesti i u neku drugu fasciklu koju je napravio korisnik, da je pročita, izbriše itd. U okviru protokola IMAP postoje komande koje korisnicima omogućavaju da prave fascikle i prebacuju poruke iz jedne svoje fascikle u drugu. Takođe, IMAP nudi komande koje korisnicima omogućavaju pretraživanje udaljenih fascikli za poruke koje ispunjavaju zadate uslove. Obratite pažnju i na to da, za razliku od POP3 servera, IMAP server između sesija prenosi informacije stanja – na primer, nazive fascikli i koje poruke mogu da se povežu sa određenom fasciklom.

Još jedna značajna osobina protokola IMAP je to što u njemu postoje komande koje omogućavaju korisnicima da preuzmu samo delove poruka. Na primer, korisnički agent može da preuzme samo zaglavlje poruke, ili samo deo višedelne MIME poruke. Ova mogućnost je izuzetno korisna kod veza malog propusnog opsega (na primer, spore modemske veze) između korisničkog agenta i njegovog servera za e-poštu. Sa ovakvim vezama moguće je da korisnik ne želi da preuzima čitave poruke u svoje poštansko sanduče, naročito ne dugačke poruke koje mogu da sadrže, na primer, zvučne ili video zapise.

Elektronska pošta zasnovana na vebu

Svakim danom sve više korisnika svoju e-poštu šalje i prima, koristeći svoje veb pretraživače. Ovu novinu uvela je kompanija *Hotmail* sredinom devedesetih, a danas je nude i *Google, Yahoo* ! kao i većina univerziteta i većih kompanija. Kod ove

usluge korisnički agent je običan veb pretraživač, a korisnik sa svojim udaljenim poštanskim sandučetom komunicira preko protokola HTTP. Kada primalac, kao što je Bob, želi da pristupi e-porukama iz svog poštanskog sandučeta, poruke se sa njegovog servera za e-poštu do njegovog pretraživača, umesto protokolima POP3 ili IMAP, šalju protokolom HTTP. Kada pošiljalac, kao što je Alisa, želi da pošalje e-poštu, poruka se iz njenog pretraživača šalje serveru za e-poštu, takođe protokolom HTTP, a ne protokolom SMTP. Međutim, Alisin server za e-poštu i dalje šalje poruke na druge servere i prima poruke sa njih, koristeći protokol SMTP.

2.5 DNS – usluga adresara interneta

Ljudi se mogu identifikovati na mnogo načina. Na primer, identifikujemo se po imenima na našim krštenicama, zatim na osnovu jedinstvenih matičnih brojeva ili, recimo, na osnovu brojeva registarskih tablica naših automobila. Mada svi ovi znaci raspoznavanja mogu da se koriste za identifikovanje ljudi, u određenim okolnostima neki od ovih identifikatora je podesniji od drugog. Na primer, računari u agenciji IRS (ne baš omiljena poreska služba u SAD) više vole brojeve osiguranja građana koji su fiksne dužine, nego njihova imena na krštenicama. Nasuprot tome, obični ljudi više vole lična imena od raznih identifikacionih brojeva. (Možete li da zamislite da neko kaže: „Zdravo, ja sam 132-67-9875. Ovo je moj suprug 178-87-1146.")

Poput ljudi i računari na internetu mogu da se identifikuju na više načina. Jedan od načina za njihovo identifikovanje predstavljaju **imena računara**. Imena računara – kao što su `cnn.com`, `www.yahoo.com`, `gaia.cs.umass.edu` i `cis.poly.edu` – lako se pamte, tako da im ljudi daju prednost. Međutim, imena računara pružaju veoma šture, ili gotovo nikakve informacije o tome gde se određeni računar nalazi unutar interneta. (Ime računara kao što je: `www.eurecom.fr` koje se završava oznakom zemlje `.fr` govori nam samo to da se odgovarajući računar verovatno nalazi u Francuskoj i ništa više.) Osim toga, budući da imena računara mogu da se sastoje od alfanumeričkih znakova promenljive dužine, ruteri bi imali velike teškoće prilikom njihove obrade. Upravo zato, računari se identifikuju i **IP adresama**.

O IP adresama podrobnije govorimo u poglavlju 4, a za sada ćemo reći samo nekoliko korisnih informacija. IP adresa se sastoji od četiri bajta i ima striktnu hijerarhijsku strukturu. Uobičajena IP adresa izgleda poput: `121.7.106.83`, gde su tačkama razdvojeni bajtovi, izraženi decimalnim brojevima od 0 do 255. IP adrese su hijerarhijske, jer ako je čitamo sleva udesno, dolazimo do sve preciznijih informacija o tome gde se određeni računar nalazi u okviru interneta (to jest, u kojoj je mreži, u okviru mreže svih mreža). Na isti način, čitajući poštansku adresu odozdo naviše, stižemo do sve preciznijih informacija o tome gde se adrese nalaze.

2.5.1 Usluge koje obezbeđuje DNS

Upravo smo videli da postoje dva načina za identifikovanje računara – imenom računara i IP adresom. Ljudi više vole, lakša za pamćenje imena računara, dok ruterima više odgovaraju hijerarhijski uređene IP adrese nepromenljive dužine. Kako bi se pomirili ovi zahtevi, neophodna nam je usluga imenika koja imena računara prevodi u IP adrese. Upravo to je i osnovni zadatak internetove usluge **DNS** (domain name system – sistem imenovanja domena). DNS je: (1) distribuirana baza podataka koja se realizuje preko hijerarhije **DNS servera** i (2) protokol aplikativnog sloja koji računarima omogućava slanje upita toj distribuiranoj bazi podataka. DNS serveri obično su UNIX računari koji koriste softver Berkeley Internet Name Domain (BIND) [BIND 2012]. Protokol DNS radi preko protokola UDP i koristi port 53.

DNS uslugu najčešće koriste ostali protokoli aplikativnog sloja – HTTP, SMTP i FTP – i to za prevođenje imena računara koje daju korisnici u IP adrese. Kao primer razmotrićemo šta se dešava kada veb pretraživač (odnosno HTTP klijent) koji se izvršava na računaru nekog korisnika zatraži objekat čiji je URL `www.nekaskola.edu/index.html`. Da bi mogao da pošalje HTTP poruku zahteva veb serveru `www.nekaskola.edu`, korisnički računar najpre mora da dođe do IP adrese računara `www.nekaskola.edu`. To se obavlja na sledeći način:

1. Na računaru ovog korisnika istovremeno se izvršava i klijentska strana DNS aplikacije.
2. Ptretraživač iz URL adrese izvlači ime računara `www.nekaskola.edu` i predaje ga klijentskoj strani DNS aplikacije.
3. DNS klijent šalje upit koji sadrži ime računara na DNS server.
4. DNS klijent na kraju u odgovoru dobija traženu IP adresu računara datog imena.
5. Kada pretraživač dobije IP adresu od DNS servera, uspostavlja TCP vezu sa HTTP serverskim procesom, smeštenim na portu 80 na toj IP adresi.

Iz ovog primera vidimo da DNS usluga uvodi dodatno – ponekad i prilično značajno – kašnjenje u sve internet aplikacije koje je koriste. Srećom, kao što ćemo ubrzo videti, tražena IP adresa najčešće se nalazi u keš memoriji „obližnjeg" DNS servera, što i te kako pomaže u smanjivanju intenziteta DNS saobraćaja i skraćivanju prosečnog DNS kašnjenja.

Pored prevođenja imena računara u IP adrese, DNS nudi još nekoliko važnih usluga:

- **Dodeljivanje pseudonima** (engl. *host aliasing*). Računar složenog imena može imati jedan pseudonim ili više njih. Na primer, ime računara `relay1.west-coast.enterprise.com` može da ima, recimo, dva pseudonima, kao što su `enterprise.com` i `www.enterprise.com`. U ovom slučaju, za ime računara `relay1.west-coast.enterprise.com` kaže se da predstavlja **zvanično ime računara** (eng. canonical hostname). Pseudonimi imena računara, ukoliko postoje, obično se lakše pamte od zvaničnog imena. DNS uslugu može da pozove aplikacija, kako bi dobila zvanično ime računara, ili njegovu IP adresu na osnovu pseudonima.

PRINCIPI U PRAKSI

DNS: NAJVAŽNIJA MREŽNA FUNKCIJA U KLIJENT-SERVER PARADIGMI

Slično protokolima HTTP, FTP i SMTP i protokol DNS je protokol aplikativnog sloja, pošto se: (1) izvršava između krajnjih sistema koji međusobno komuniciraju, koristeći klijentsko serversku paradigmu i (2) oslanja na istaknuti transportni protokol od jednog do drugog kraja za prenos DNS poruka između krajnjih sistema koji komuniciraju. Međutim, s druge strane, uloga protokola DNS prilično se razlikuje od aplikacija za veb, prenos datoteka i e-pošte. Za razliku od tih aplikacija, DNS nije aplikacija koju korisnici neposredno koriste. Umesto toga, DNS nudi jednu od najvažnijih internet funkcija – tačnije, prevođenje imena računara u njihove IP adrese za korisničke aplikacije i ostali softver na internetu. U poglavlju 1.2 smo napomenuli da se većina složenosti internet arhitekture nalazi po „obodima" mreže. DNS, koji realizuje važan proces prevođenja imena u adrese, koristeći klijente i servere koji se nalaze na obodima mreže, još jedan je primer ovako dizajnirane filozofije.

- **Dodeljivanje pseudonima serverima za e-poštu** (eng. mail server aliasing). Iz sasvim očiglednih razloga bitno je da adrese elektronske pošte budu jednostavne za pamćenje. Ukoliko bi, na primer, Bob imao korisnički nalog na *Hotmail*-u, njegova e-adresa mogla bi da bude jednostavno: `bob@hotmail.com`. Međutim ime *Hotmail*-ovog servera za e-poštu mnogo je složeniji i mnogo teži za pamćenje od jednostavnog `hotmail.com` (zvanično ime moglo bi, recimo, da glasi: `relay1.west-coast.hotmail.com`). Aplikacija za e-poštu može da iskoristi DNS uslugu za dobijanje zvaničnog imena i IP adrese računara čiji pseudonim ima. U stvari, zapis MX (pogledaj ispod) omogućava da server za e-poštu i veb server neke kompanije imaju ista imena (pseudonime); na primer, oba servera neke kompanije mogu da se zovu: `kompanija.com`.
- **Raspodela opterećenja.** DNS usluga se takođe koristi za raspodeljivanje opterećenja između nekoliko istovetnih servera, što je čest slučaj kod repliciranih veb servera. Veoma posećene lokacije, kao što je, recimo `cnn.com`, imaju replike na nekoliko servera, od kojih se svaka izvršava na drugom krajnjem sistemu sa drugačijom IP adresom. Kod repliciranih veb servera, *skup* IP adresa se vezuje za jedno zvanično ime. Ovaj skup IP adresa se nalazi u bazi podataka DNS servera. Kada klijent postavi DNS upit za ime, preslikan na ovaj skup adresa, server odgovara čitavim skupom IP adresa, s tim što u svakom svom odgovoru menja njihov redosled. Pošto klijent svoju HTTP poruku zahteva obično šalje na prvu IP adresu na spisku, DNS usluga na ovaj način obezbeđuje raspodelu saobraćaja između replika servera. DNS rotacija se koristi i za e-poštu, tako da veći broj servera za e-poštu može da ima isti pseudonim. Odnedavno kompanije koje obezbeđuju distribuciju sadržaja na internetu, među kojima i *Akamai*, sofisticiranim korišćenjem DNS usluge [Dilley 2002], ostvaruju distribuciju veb sadržaja (pročitajte poglavlje 7).

DNS je definisan u dokumentima RFC 1034 i RFC 1035, a najnovije verzije se nalaze u još nekoliko dodatnih RFC dokumenata. Reč je o veoma složenom sistemu čije smo ključne aspekte samo dotakli na ovom mestu. Zainteresovani čitalac se upućuje na RFC dokumenta i knjigu koju su napisali Albitz i Liu [Albitz 1993]; videti takođe retrospektivu u radu [Mockapetris 1988] u kome je dat sjajan opis o tome šta je i kako radi DNS, kao i radu [Mockapetris 2005].

2.5.2 Prikaz načina rada DNS usluge

Sada ćemo opširnije prikazati kako radi DNS. U izlaganju koje sledi usredsredićemo se na uslugu prevođenja imena računara u IP adrese.

Pretpostavimo da bi neka aplikacija (recimo, veb pretraživač ili program za elektronsku poštu) koja se izvršava na računaru nekog korisnika trebalo da prevede ime računara u IP adresu. Ova aplikacija najpre poziva klijentsku stranu DNS usluge, navodeći joj ime računara koji bi treba da se prevede. (Na većini računara pod UNIX operativnim sistemom proces prevođenja se pokreće tako što aplikacija poziva funkcijuget `host by name`). Zatim DNS na korisnikovom računaru preuzima inicijativu i šalje poruku upita na mrežu. Sve DNS poruke upita i odgovora šalju se u okviru UDP datagrama na port 53. Nakon kašnjenja, od nekoliko milisekundi do nekoliko sekundi, DNS na korisnikovom računaru prima DNS poruku odgovora u kojoj je navedeno traženo preslikavanje. Ovo preslikavanje prosleđuje se aplikaciji koja ga je zatražila. Dakle, sa stanovišta aplikacije koja ga poziva, DNS je crna kutija koja obezbeđuje jednostavnu, jasnu uslugu prevođenja. Međutim, ova crna kutija je, u stvari, veoma složena i obuhvata veliki broj DNS servera koji su raspoređeni širom sveta, kao i protokol aplikativnog sloja kojim se tačno utvrđuje komunikacija između DNS servera i računara koji šalju upite.

U pojednostavljenoj varijanti, DNS uslugu činio bi jedan DNS server u kome bi se nalazila sva preslikavanja. U ovakvom centralizovanom rešenju klijenti bi sve upite upućivali jednom DNS serveru koji bi direktno odgovarao svim klijentima. Iako jednostavnost ovakvog rešenja izgleda privlačno, ona je nepodesna za potrebe savremenog interneta sa neizmernim (i sve većim) brojem računara. Problemi sa centralizovanim dizajnom navedeni su u produžetku:

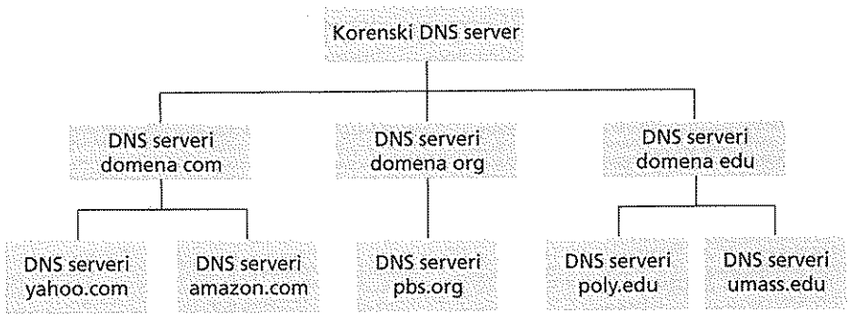
- **Mreža se oslanja na jednu tačku.** U slučaju kvara DNS servera, internet postaje neupotrebljiv.
- **Intenzitet saobraćaja.** Jedan DNS server morao bi da se nosi sa svim DNS upitima (za sve HTTP zahteve i poruke e-pošte stotina miliona računara).

- **Udaljena centralizovana baza podataka.** Jedan DNS server ne bi mogao da bude „blizu" svih klijenata koji šalju upite. Ukoliko bismo postavili jedan DNS server u Njujorku, svi upiti iz Australije morali bi da putuju na drugi kraj sveta – možda sporim i zagušenim vezama. Kašnjenja bi u tom slučaju bila zaista značajna.
- **Održavanje.** Jedan DNS server morao bi da čuva zapise za sve računare na internetu. Ovakva centralizovana baza podataka, ne samo da bi bila ogromna, već bi veoma često morala da se ažurira za svaki novi računar u mreži.

Ukratko, centralizovana baza podataka na jednom DNS serveru jednostavno *nije moguća*. Zato je usluga DNS napravljena tako da bude distribuirana. U stvari, DNS je izuzetno lep primer kako se distribuirana baza podataka može primeniti na internetu.

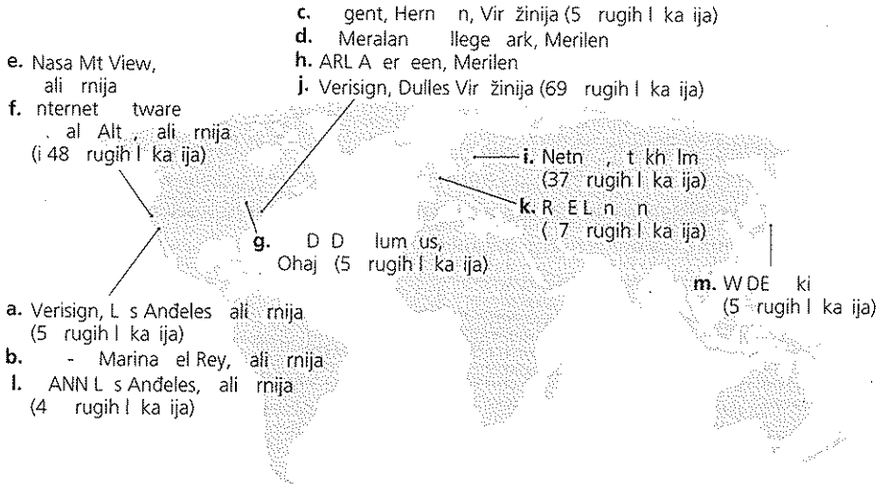
Distribuirana, hijerarhijska baza podataka

Kako bi se izborio sa svim zahtevima, DNS koristi veliki broj servera koji su hijerarhijski organizovani i raspoređeni širom sveta. Nijedan od DNS servera ne može da izvrši preslikavanja za sve računare na internetu. Umesto toga, preslikavanje se raspodeljuje između DNS servera. Uopšteno govoreći, postoje tri klase DNS servera – korenski (eng. root) DNS serveri, DNS serveri domena najvišeg nivoa (eng. top-level domain, TLD) i nadležni DNS serveri – organizovani u hijerarhiju koja je prikazana na slici 2.19. Da biste razumeli međusobni odnos ove tri vrste servera, pretpostavimo da je DNS klijentu potrebna IP adresa računara hosta `www.amazon.com`. Za ovu prvu pretpostavku, opisaćemo događaje koji će se desiti. Klijent najpre stupa u vezu sa nekim od korenskih DNS servera koji mu vraćaju IP adrese TLD servera za najviši domen `com`. Klijent se zatim obraća jednom od ovih TLD servera koji mu vraća IP adresu nadležnih servera za `amazon.com`. Konačno, klijent kontaktira jedan od nadležnih servera za `amazon.com` od kojih dobija IP adresu računara čiji je ime: `www.amazon.com`. Uskoro ćemo ovaj postupak DNS pretraživanja prikazati podrobnije, ali pre toga razmotrimo malo bolje ove tri klase DNS servera:



Slika 2.19 ◆ Deo hijerarhije DNS servera

- **Korenski DNS serveri.** Na internetu postoji trinaest korenskihDNS servera (imaju oznake od A do M), od kojih se većina nalazi u Severnoj Americi. Na slici 2.20 možete da vidite mapu korenskihDNS servera iz oktobra 2006. godine, spisak trenutnih korenskih DNS servera možete da potražite preko [Root-servers 2012]. Iako smo svih trinaest korenskihDNS servera označili kao da je u pitanju jedan računar, svaki „server" u stvari predstavlja mrežu replicaranihservera kojapruža veću bezbednost i pouzdanost. Sveukupno, u jesen 2011. godine bilo je 247 korenskih servera.



Slika 2.20 ◆ Korenski DNS serveri u 2012. godini (naziv, organizacija, lokacija)

- **Serveri domena najvišeg nivoa (TLD serveri).** Ovi serveri odgovorni su za domene najvišeg nivoa, kao što su domeni com, org, net, edu i gov, kao i za sve domene najvišeg nivoa raznih zemalja kao što su uk, fr, ca i jp. Kompanija Verisign Global Registry Services vodi računa o TLD serverima za domen najvišeg nivoa com, dok je kompanija Educause zadužena za TLD servere domena najvišeg nivoa edu. Videti [IANA TLD 2012] za spisak svih domena najvišeg nivoa.
- **Nadležni DNS serveri.** Svaka organizacija koja ima javno dostupne računare (recimo, veb servere ili servere za e-poštu) na internetu, mora da obezbedi javno dostupne DNS zapise u kojima se imenaovih računara preslikavaju u IP adrese. Ovi DNS zapisi čuvaju se na nadležnim DNS serverima tih organizacija. Organizacija može da ima sopstveni nadležan DNS server na kome čuva te zapise ili može da zakupi takav DNS server nekog drugog posrednika za internet usluge.

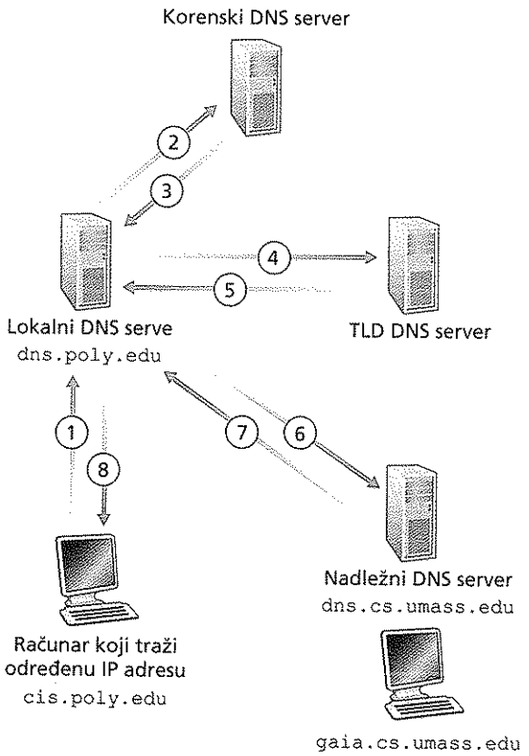
Većina univerziteta i velikih kompanija obično koristi i održava sopstvene primarne i sekundarne (rezervne) nadležne DNS servere.

Korenski, TLD serveri i nadležni DNS serveri sačinjavaju hijerarhiju DNS servera prikazanu na slici 2.19. Međutim, postoji još jedan veoma važan tip DNS servera koji se zovu **lokalni DNS serveri**. Lokalni DNS serveri se ne uklapaju sasvim u navedenu hijerarhiju servera, ali bez sumnje zauzimaju vrlo važno mesto u DNS arhitekturi. Svi posrednici za internet usluge (ISP) – kao što su univerziteti, akademske ustanove, kompanije, ili ISP za stanovništvo – imaju svoje lokalne DNS servere (zovu se i podrazumevani serveri imena). Kada se neki računar poveže sa posrednikom za internet usluge, on od njega dobija IP adrese jednog ili više njegovih lokalnih DNS servera (obično koristeći protokol DHCP o kome govorimo u poglavlju 4). IP adresu svog lokalnog DNS servera možete da vidite otvaranjem prozora za praćenje statusa mreže u operativnim sistemima *Windows* ili *UNIX*. Lokalni DNS server računara obično se nalazi „blizu" tog računara. Kod ISP-a za institucije lokalni DNS server može da bude u istoj lokalnoj mreži kao i računar host; dok kod ISP-a za stanovništvo, DNS i računar razdvaja svega nekoliko rutera. Kada računar napravi DNS upit, taj upit se šalje lokalnom DNS serveru, koji se ponaša kao proksi server i prosleđuje ga u hijerarhiju DNS servera, o čemu mnogo podrobnije govorimo u produžetku.

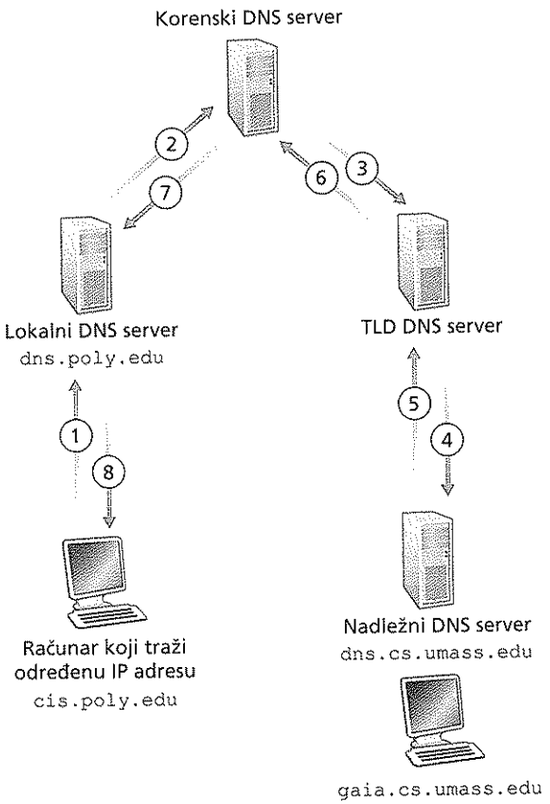
Pređimo sada na jednostavan primer. Pretpostavimo da je računaru `cis.poly.edu` potrebna IP adresa računara `gaia.cs.umass.edu`. Takođe, pretpostavimo i to da se lokalni DNS server Politehničkog univerziteta zove `dns.poly.edu`, a da se nadležni DNS server računara `gaia.cs.umass.edu` zove `dns.umas.edu`. Kao što je prikazano na slici 2.21, računar `cis.poly.edu` najpre šalje DNS poruku upita svom lokalnom serveru `dns.poly.edu`. Poruka upita sadrži ime računara, u ovom slučaju `gaia.cs.umass.edu`, koji bi trebalo da se prevede. Lokalni DNS server prosleđuje ovu poruku upita korenskom DNS serveru koji, videvši sufiks edu, vraća lokalnom DNS serveru spisak IP adresa TLD servera odgovornih za domen `edu`. Lokalni DNS server zatim ponovo šalje poruku upita jednom od ovih TLD servera. TLD server, videvši sufiks `umass.edu`, odgovara IP adresom nadležnog DNS servera na Univerzitetu u Masačusetsu, tačnije, `dns.umass.edu`. Konačno, lokalni DNS server ponovo šalje poruku upita direktno serveru `dns.umass.edu`, od koga dobija IP adresu računara `gaia.cs.umass.edu`. Obratite pažnju na to da je za preslikavanje imena jednog računara poslato čak osam DNS poruka: četiri poruke upita i četiri poruke odgovora! Uskoro ćemo videti na koji način to može da se smanji DNS keširanjem.

U ovom primeru pretpostavili smo da TLD server zna na kom nadležnom DNS serveru se nalazi traženo ime računara. U većini slučajeva ovo nije tačno. Umesto toga, TLD server zna samo posredni DNS server, koji sa svoje strane zna na kom nadležnom DNS serveru se nalazi traženi podatak za dato ime računara. Na primer,

pretpostavimo ponovo da Univerzitet u Masačusetsu ima DNS server za čitav univerzitet koji se zove `dns.umass.edu`. Pretpostavimo i to da svaki odsek ovog univerziteta ima sopstveni DNS server i da je svaki od njih nadležan za sve računare u svom odseku. U ovom slučaju, kada posredni DNS server `dns.umass.edu` primi upit za računar čije se ime završava sa `cs.umass.edu`, on DNS serveru `dns.poly.edu` vraća IP adresu DNS servera `dns.cs.umass.edu` koji je nadležan za računare čija se imena završavaju sa `cs.umass.edu`. Lokalni DNS server `dns.poly.edu` zatim šalje upit nadležnom DNS serveru, koji za uzvrat preslikava traženi računar host. U ovom slučaju poslato je ukupno deset DNS poruka!



Slika 2.21 ◆ Interakcija između različitih DNS servera



Slika 2.22 ◈ Interakcija između različitih DNS servera

U primeru prikazanom na slici 2.21 koriste se i **rekurzivni** i **iterativni upiti**. Upit poslat sa računara `cis.poly.edu` je rekurzivan upit, jer traži od servera `dns.poly.edu` da izvrši preslikavanje u njegovo ime. Međutim, preostala tri upita su iterativna pošto se svi odgovori šalju direktno serveru `dns.poly.edu`. Teoretski, svaki DNS upit može da bude i rekurzivan i iterativan. Na primer, slika 2.22 prikazuje lanac DNS upita u kome su svi ovi upiti rekurzivni. U praksi, upiti obično prate šablon sa slike 2.21: upit od računara koji traži neku IP adresu do lokalnog DNS servera je rekurzivan, dok su ostali upiti iterativni.

DNS keširanje

U dosadašnjem razmatranju zanemarivali smo **DNS keširanje**, inače veoma važnu funkciju DNS sistema. U stvari, DNS sistem naširoko koristi DNS keširanje, kako bi se poboljšale karatkeristike vezane za kašnjenja i smanjio broj DNS poruka koje se šire internetom. Ideja u osnovi DNS keširanja krajnje je jednostavna. Kada u lancu upita DNS server primi DNS odgovor (koji sadrži, recimo, preslikavanje imena nekog računara u IP adresu), server ovo preslikavanje privremeno smešta (kešira) u svoju lokalnu memoriju. Na primer, na slici 2.21, svaki put kada lokalni DNS server `dns.poly.edu` primi odgovor od nekog DNS servera u lancu, može da kešira bilo koju od informacija, koje se nalaze u tom odgovoru. Ukoliko je par za ime računara/IP adresa keširan, DNS server može da obezbedi traženu IP adresu, čak i ako nije nadležan za to ime računara. DNS server briše keširanu informaciju nakon isteka određenog vremena (najčešće je podešeno dva dana), pošto računari i preslikavanje imena računara u IP adrese nisu trajni.

Na primer, pretpostavimo da računar `apricot.poly.edu` serveru `dns.poly.edu` pošalje upit, tražeći IP adresu računara imena `cnn.com`. Dalje, pretpostavimo i to da je nekoliko sati kasnije neki drugi računar sa Politehničkog univerziteta, recimo kiwi.poly.fr, serveru `dns.poly.edu` takođe poslao isti upit. Zahvaljujući keširanju, lokalni DNS server može odmah da vrati IP adresu računara `cnn.com` ovom drugom računaru koji šalje upit, pri čemu ne mora da šalje upit drugim DNS serverima. Lokalni DNS serveri takođe mogu da keširaju IP adrese TLD servera, čime se dozvoljava lokalnim DNS serverima da u lancu upita preskoče korenske DNS servere (ovo se često dešava).

2.5.3 DNS zapisi i poruke

DNS serveri, koji zajedno realizuju distribuiranu DNS bazu podataka, čuvaju **zapise o resursu** (Resource Record, RR), uključujući i zapise koji omogućavaju preslikavanje imena računara u njihove IP adrese. U svakoj DNS poruci odgovora nalazi se jedan ili više zapisa o resursu. U ovom i sledećem odeljku nalazi se sažet prikaz DNS zapisa o resursu i poruka; podrobnije informacije možete pronaći u knjizi [Abitz 1993], ili u RFC dokumentima koji se odnose na uslugu DNS [RFC 1034; RFC 1035].

Zapis o resursu ima sledeća četiri polja:

`(Name, Value, Type, TTL)`

Polje `TTL` predstavlja rok trajanja zapisa resursa, vreme života; ovo polje određuje kada se određeni zapis resursa uklanja iz keš memorije. U primerima zapisa datim ispod zanemarili smo polje TTL. Značenja polja `Name` i `Value` zavise od polja `Type`:

- Ukoliko je `Type=A`, onda je polje `Name` ime računara, a `Value` njegova IP adresa. Dakle, zapisom tipa A nudi se uobičajeno preslikavanje imena računara u njegovu IP adresu. Na primer, `(relay1.bar.foo.com, 145.37.93.126, A)` je zapis tipa A.
- Ukoliko je `Type=NS`, onda je polje `Name` domen (na primer, `foo.com`), a `Value` ime računara nadležnog DNS servera koji može da obezbedi IP adrese

računara u tom domenu. Ovaj zapis se koristi za preusmeravanje DNS upita u lancu upita. Na primer, (`foo.com, dns.foo.com, NS`) je zapis tipa NS.

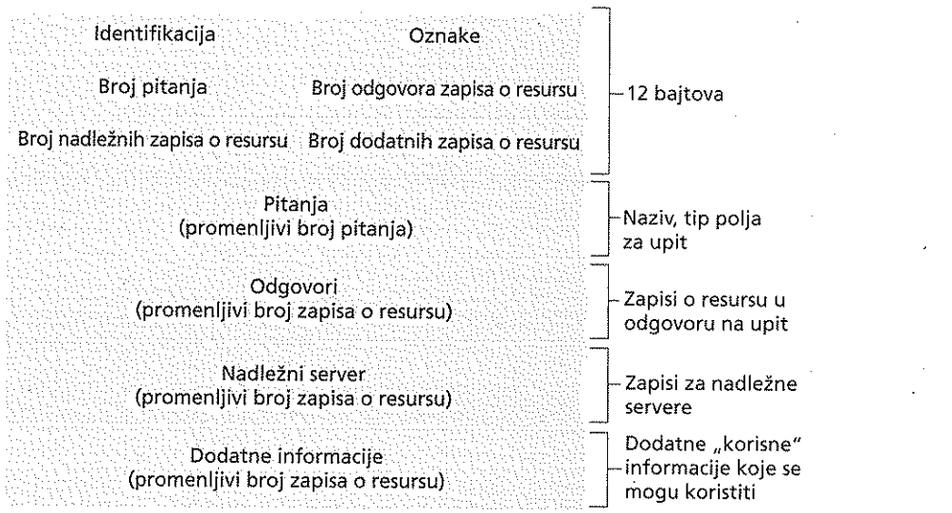
- Ukoliko je `Type=CNAME`, onda je polje `Value` zvanično ime računara čiji je pseudonim `Name`. Pomoću ovog zapisa računari koji traže upit mogu da saznaju zvanično ime računara. Na primer, (`foo.com, relay1.bar.foo.com, CNAME`) je zapis tipa CNAME.
- Ukoliko je `Type=MX`, onda je polje `Value` zvanično ime servera e-pošte čiji je pseudonim `Name`. Na primer, (`foo.com. mail.bar.foo.com, MX`) je zapis tipa MX. Ovaj tip zapisa omogućava računarima servera e-pošte da imaju jednostavne pseudonime. Obratite pažnju na to da korišćenje MX zapisa omogućava kompanijama da koriste isti pseudonim za svoj server e-pošte i za neki drugi server (recimo, veb server). Da bi dobio zvanično ime servera e-pošte, DNS klijent upitom traži MX zapis, a kada mu je potrebno zvanično ime nekog drugog servera, klijent traži zapis tipa CNAME.

Ukoliko je neki DNS server nadležan za određeno ime računara, tada taj DNS server ima zapis tipa A za to ime računara. (Čak i kada nije nadležan, DNS server može u svojoj keš memoriji da sadrži zapis tipa A.) Ukoliko server nije nadležan za ime računara, tada će server sadržati zapis Type NS za domen koji sadrži ime računara; on takođe sadrži i zapis tipa A koji obezbeđuje IP adresu DNS servera u polju `Value` zapisa NS. Na primer, pretpostavimo da TLD server domena edu nije nadležan za računar host`gaia.cs.umass.edu`. U tom slučaju ovaj server sadrži zapis za domen u kome se nalazi računar host `gaia.cs.umass.edu`, na primer (`umass.edu, dns.umass.edu, NS`). TLD server domena edu sadrži i zapis tipa A koji preslikava ime DNS servera `dns.umass.edu` u IP adresu, na primer (`dns.umass.edu, 128.119.40.111, A`).

DNS poruke

Ranije u ovom odeljku pomenuli smo DNS poruke upita i odgovora. Ovo su ujedno i jedine vrste DNS poruka. Štaviše, kao što je prikazano na slici 2.23, poruke upita i odgovora imaju isti format. U nastavku objašnjavamo značenje različitih polja DNS poruka.

- Prvih 12 bajtova predstavljaju *odeljak zaglavlja*, koji ima nekoliko polja. Prvo polje je 16-bitni broj koji identifikuje upit. Ovaj identifikator se kopira u poruku odgovora, što omogućava klijentu da poredi primljene odgovore sa poslatim upitima. U polju za oznake koristi se nekoliko oznaka. Jednobitna oznaka tipa upit/odgovor naznačava da li je poruka upit (0) ili odgovor (1). Jednobitna oznaka nadležnosti postavlja se u poruku odgovora ukoliko je određeni DNS server nadležan za ime iz upita. Jednobitna oznaka za traženje rekurzije se postavlja kada klijent (računar host ili DNS server) želi da odgovarajući DNS server izvrši rekurziju, ukoliko nema zapis. Ako DNS server podržava rekurziju, jednobitna oznaka odgovarajućeg polja se postavlja u odgovoru. U zaglavlju postoje i četiri polja sa brojevima kojima se naznačava koliko su se puta ponovila četiri tipa odeljaka podataka koji slede iza zaglavlja.



Slika 2.23 ◈ Format DNS poruke

- *Odeljak za pitanja* sadrži informacije o upitu koji se pravi. U ovom odeljku nalaze se: (1) polje sa imenom koje sadrži ime za koje se traži upit i (2) polje za tip koje sadrži tip pitanja koje se postavlja vezano za ime – na primer, povezivanje adrese računara i imena (tip A), ili servera za e-poštu za ime (tip MX).
- U odgovoru DNS servera *odeljak za odgovor* sadrži zapise o resursu za ime za koje se upit prvobitno tražio. Sećate se da svaki zapis resursa ima polja: `Type` (na primer, A, NS, CNAME ili MX), `Value` i TTL. U odgovoru može da se nađe više zapisa o resursu, zato što traženo ime računara može da ima više IP adresa (na primer, za replike veb servera, o čemu smo govorili ranije u ovom odeljku).
- *Odeljak za nadležnost* sadrži zapise o ostalim nadležnim serverima.
- U *dodatnom odeljku* nalaze se ostali korisni zapisi. Na primer, u odeljku za odgovor u povratnoj poruci na MX upit nalazi se zapis o resursu u kome je navedeno zvanično ime računara servera za e-poštu. U dodatnom odeljku se nalazi zapis o resursu tipa A, u kome je navedena IP adresa za zvanično ime računara servera za e-poštu.

Da li biste želeli da DNS poruku upita pošaljete direktno sa računara na kome radite nekom DNS serveru? Ovo može da se uradi prilično jednostavno, ukoliko

koristite **program nslookup** koji je dostupan na većini *Windows* i *UNIX* platformi. Na primer, na računaru sa operativnim sistemom *Windows* otvorite komandnu liniju i jednostavnim upisivanjem „nslookup" pozovite ovaj program. Kada se program pozove, možete da pošaljete DNS upit bilo kom DNS serveru (korenskom, TLD ili nadležnom). Kada od DNS servera primi poruku odgovora, program *nslookup* prikazuje zapise koji su dobijeni u odgovoru (u čitljivom obliku za korisnike). Osim što ovaj program možete da pokrenete sa svog računara, isto to možete da učinite sa neke od mnogih veb lokacija koje omogućavaju njegovo daljinsko korišćenje. (U bilo kom pretraživaču upišite „nslookup" i naći ćete se na nekoj od ovih lokacija.) DNS *Wireshark* laboratorijska vežbanja na kraju ovog poglavlja dozvoljavaju vam da istražite DNS mnogo detaljnije.

Unošenje zapisa u DNS bazu podataka

U dosadašnjem izlaganju govorili smo samo o tome kako se zapisi preuzimaju iz DNS baze podataka. Možda se pitate kako te informacije uopšte dospevaju u nju. Pogledajmo kako to izgleda na određenom primeru. Pretpostavimo da ste upravo osnovali kompaniju po imenu „Network Utopia". Prvo što ćete sasvim sigurno želeti da učinite je da registrujete naziv domena `networkutopia.com` u registratoru domena. **Registrator domena** je firma koja proverava jedinstvenost imena domena, unosi ta imena u DNS bazu podataka (uskoro ćete videti kako) i za sve svoje usluge dobija malu nadoknadu. Do 1999. godine jedini registrator domena, kompanija *Network Solutions*, imala je monopol na registraciju domena `com`, `net` i `org`. Međutim, danas postoji mnogo registratora domena koji se bore da pridobiju korisnike, a sve njih akredituje udruženje ICANN (Internet Corporation for Assigned Names and Numbers). Spisak registratora domena možete da potražite na adresi `http://www.internic.net`.

Prilikom prijavljivanja domena `networkutopia.com` kod nekog registratora domena potrebno je da mu date imenai IP adrese primarnog i sekundarnog nadležnog DNS servera. Pretpostavimo da su nazivi i IP adrese `dns1.networkutopia.com`, `dns2.networkutopia.com` `212.212.212.1` i `212.212.212.2`. Za svaki od ova dva nadležna DNS servera registrator domena unosi zapis o resursu tipa NS i A na TLD servere za domen com. Tačnije, za primarni nadležni server domena `networkutopia.com` registrator unosi sledeća dva zapisa o resursu u DNS sistem:

```
(networkutopia.com, dns1.networkutopia.com, NS)

(dns1.networkutopia.com, 212.212.212.1, A)
```

Pored toga, neophodno je i da se zapis o resursu tipa A za veb server `www.networkutopia.com` i zapis o resursu tipa MX za server za e-poštu `mail.networkutopia.com` unesu na nadležne DNS servere. (Sve donedavno sadržaj svakog DNS servera konfigurisao se statički, odnosno konfiguracionom datotekom koju bi

RANJIVOST DNS SISTEMA

Videli smo da je DNS jedan od najvažnijih delova internet infrastrukture i da obezbeđuje mnoge važne usluge – uključujući veb i e-poštu – koje jednostavno ne bi mogle da rade bez njega. Prirodno se postavlja pitanje, može li DNS da bude napadnut? Da li je DNS glineni golub, koji čeka da bude ustreljen, čijim bi padom pala i većina internet aplikacija?

Prva vrsta napada koja pada na pamet jeste DDoS napad, zagušenjem propusnog opsega (pogledajte odeljak 1.6) na DNS servere. Na primer, napadač bi mogao da korenski DNS server preplavi paketima, sa toliko mnogo paketa tako da onemogući odgovaranje na najveći deo redovnih DNS upita. Takav značajniji DDoS napad na osnovne DNS servere zaista se i dogodio 21. oktobra 2002. U tom napadu napadači su za slanje hrpe ICMP ping poruka na svaki od 13 osnovnih DNS servera koristili mrežu zloupotrebljenih računara da učestvuju u tome. (O ICMP porukama pričamo u poglavlju 4. Za sada je dovoljno da znate da su ICMP paketi posebna vrsta IP datagrama.) Srećom, ovaj veliki napad je imao zanemarljive posledice, pošto većina korisnika interneta nije ni primetila da se nešto neobično događa. Napadači jesu uspeli da paketima preplave korenske servere. Međutim, većina osnovnih DNS servera zaštićena je sistemima za filtriranje paketa, konfigurisanih tako da uvek zaustave sve ICMP ping poruke usmerene ka korenskim serverima. Ovi zaštićeni serveri su tako bili pošteđeni i radili su sasvim uobičajeno. Pored toga, većina lokalnih DNS servera kešira IP adrese servera za domene najvišeg nivoa, čime postupak slanja upita obično zaobilazi korenske DNS servere.

Potencijalno mnogo opasniji DDoS napad na DNS sistem bio bi slanje mnoštva DNS upita na servere za domene najvišeg nivoa, na primer, na sve servere najvišeg nivoa koji vode računa o domenu com. Mnogo je teže filtrirati DNS upite usmerene na ove DNS servere, a uz to servere za domene najvišeg nivoa nije moguće tako lako zaobići, kao što je slučaj sa korenskim serverima. Međutim, ozbiljnost ovakvog napada moguće je bar delimično ublažiti keširanjem u lokalnim DNS serverima.

DNS može da bude napadnut i na druge načine. U tzv. napadu čoveka u sredini, napadač presreće upit od računara hosta i vraća mu lažan odgovor. U tzv. napadu trovanjem, napadač šalje lažan odgovor DNS serveru, tako da server u svoju keš memoriju nesvesno unosi taj lažan podatak. Ove dve vrste napada mogle bi da se koriste za, na primer, preusmeravanje korisnika veba, koji ništa ne sumnja, na veb stranicu napadača. Međutim, ovi napadi se teško izvode, pošto je za njih potrebno presretati pakete ili odglumiti DNS server [Skoudis 2006].

Još jedna važna vrsta napada na DNS sistem nije napad na DNS uslugu samu po sebi, već se umesto toga iskorišćava DNS infrastruktura za pokretanje DDoS napada na računar koji je meta napada (na primer, server za e-poštu na nekom univerzitetu). U ovom napadu, napadač šalje DNS upite većini nadležnih DNS servera pri čemu svi upiti imaju lažnu izvornu adresu računara koji je meta napada. DNS serveri, zatim, šalju svoje odgovore neposredno na taj računar. Pošto se upiti mogu vešto obraditi tako da odgovor bude mnogo veći (u bajtovima) od upita (tzv. pojačanje), napadač može da pretrpa porukama cilj napada, pri čemu sâm ne povećava svoj saobraćaj. Ovakvi napadi preusmeravanjem poruka, koji koriste DNS sistem, do sada su postigli samo ograničeni uspeh [Mirkovic 2005].

Ukratko, DNS je pokazao da je iznenađujuće otporan prema napadima. Do sada nije bilo napada koji bi ozbiljnije ugrozio DNS uslugu. Bilo je uspešnijih napada umnožavanjem poruka, međutim, ovi napadi mogu se odbiti (i jesu bili odbijeni) odgovarajućim konfigurisanjem DNS servera.

pravio administrator sistema. Odskora je u protokol DNS dodata opcija UPDATE koja omogućava dinamičko dodavanje, ili brisanje podataka iz baze podataka putem DNS poruka. Ovakvo dinamičko ažuriranje opisano je u dokumentima [RFC 2136] i [RFC 3007].)

Kada uradite sve što smo ovde naveli, ljudi će moći da posete vašu veb lokaciju i pošalju elektronsku poštu zaposlenima u vašoj kompaniji. Izlaganje o DNS sistemu završićemo proverom istinitosti ovog iskaza. Ova provera je takođe korisna i da utvrdite ono što ste naučili o DNS-u. Pretpostavimo da Alisa iz Australije želi da poseti veb stranu `www.networkutopia.com`. Kao što smo već rekli, njen računar najpre šalje DNS upit njenom lokalnom DNS serveru. Ovaj lokalni DNS server se odmah zatim obraća TLD serveru za domen `com`. (Ukoliko nema keširanu adresu TLD servera za domen `com`, lokalni DNS server će najpre morati da se obrati nekom korenskom DNS serveru.) NaTLD serveru nalaze se zapisi o resursu tipa A i tipa NS, koji su izlistani gore, budući da ih je registrator već uneo u sve TLD servere za domen com. TLD server u svom odgovoru Alisinom lokalnom DNS serveru šalje dva zapisa o resursu. Nakon toga, lokalni DNS server šalje DNS upit na adresu `212.212.212.1`, tražeći zapis tipa A koji odgovara imenu `www.networkutopia.com`. U ovom zapisu nalazi se IP adresa željenog veb servera, recimo `212.212.71.4`, koju lokalni DNS server vraća Alisinom računaru. Alisin veb pretraživač sada može da uspostavi TCP vezu sa računarom `212.212.71.4` i pošalje mu HTTP zahtev. Eto, krstarenje vebom i nije toliko jednostavno, koliko na prvi pogled izgleda!

2.6 P2P aplikacije

Sve aplikacije o kojima smo do sada govorili u ovom poglavlju – uključujući veb, e-poštu i DNS – koriste klijentsko serversku arhitekturu u kojoj se značajno oslanjaju na uvek uključene infrastrukturne servere. Sećate se iz odeljka 2.1.1 da u P2P arhitekturi postoji minimalna (ili je nema) zavisnost od uvek uključenih servera. Umesto toga, par privremeno povezanih računara, koji se nazivaju ravnopravni računari (engl. *peer*), međusobno direktno komuniciraju. Ovi računari ne pripadaju posrednicima za internet usluge, već su u pitanju stoni računari ili laptopovi, kojima upravljaju korisnici.

U ovom odeljku istražujemo dve različite aplikacije koje su posebno dobro prilagođene korišćenju P2P arhitekture. Prva aplikacija se koristi za distribuciju datoteka, pri čemu se datoteke sa jednog računara distribuiraju do većeg broja ravnopravnih računara. Distribucija datoteka je sjajno mesto za početak istraživanja P2P aplikacija, pošto se iz nje jasno vidi prilagodljivost P2P arhitekture. Kao primer aplikacije za distribuciju datoteka, opisujemo popularni BitTorrent sistem. Druga P2P aplikacija koju istražujemo je distribuirane baze podataka u velikoj zajednici

ravnopravnih računara. Za ovu aplikaciju, istražićemo koncept distribuirane heš tabele (DHT – Distributed Hash Table).

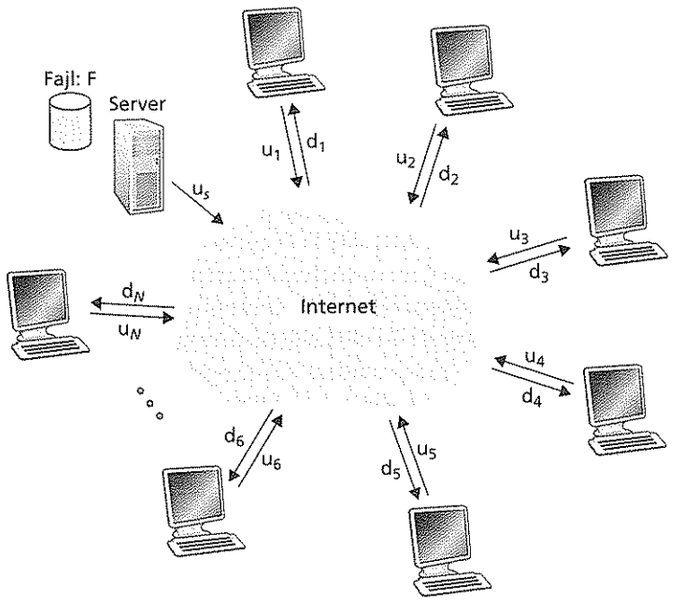
2.6.1 P2P distribucija datoteka

Putovanje u svet P2P aplikacija počinjemo onom koja se nameće kao prirodan izbor, tačnije, distribuiranje velike datoteke od jednog servera na veći broj računara (nazvanih ravnopravni računari). Ta datoteka može da bude nova verzija operativnog sistema *Linux*, softverska zakrpa za postojeći operativni sistem ili aplikaciju, MP3 muzička datoteka, ili MPEG video datoteka. Klijentsko-serverska distribucija datoteka znači da server mora da pošalje kopiju određene datoteke svim ravnopravnim računarima – što za taj server može da bude izuzetno veliko opterećenje i da znači zauzimanje velikog dela propusne moći tog servera. Pri P2P distribuciji datoteka, svi ravnopravni računari koji učestvuju u distribuiranju mogu dalje da prenesu deo datoteke, koju su primili od drugih ravnopravnih računara, na taj način pomažući serveru u procesu distribucije. Od 2012. godine, najpopularniji P2P protokol za distribuciju datoteka je BitTorrent. Tvorac izvornog protokola BitTorrent je Bram Kohen, a sada postoje brojni različiti nezavisni BitTorrent klijenti koji poštuju pravila protokola BitTorrent, baš kao što postoje brojni klijenti veb pretraživača koji poštuju pravila protokola HTTP. U ovom pododeljku prvo istražujemo prilagodljivost P2P arhitekture u kontekstu distribucije datoteka. Zatim podrobnije opisujemo BitTorrent , ističući njegove najvažnije karakteristike i osobine.

Prilagodljivost P2P arhitekture

Da bismo poredili klijentsko-serversku arhitekturu sa P2P arhitekturom i pokazali prirodnu prilagodljivost svojstvenu P2P arhitekturi, razmotrićemo jednostavan kvantitativni model distribuiranja datoteka do istog broja ravnopravnih računara za obe vrste arhitekture. Kao što je prikazano na slici 2.24, server i ravnopravni računari su povezani sa internetom preko pristupnih linkova. Označimo uzvodnu brzinu pristupnog linka servera sa u_s, uzvodnu brzinu pristupnog linka i-tog računara sa u_i, a brzinu preuzimanja pristupnog linka i-tog računara sa d_i. Takođe, veličinu datoteke koja bi trebalo da se distribuira (u bitovima) označimo sa F, a broj ravnopravnih računara koji žele da dobiju kopiju te datoteke označimo sa N. **Vreme distribucije** je vreme koje je potrebno da svih N ravnopravnih računara dobiju kopiju datoteke. U analizi vremena distribucije koje sledi, i za klijentsko-serversku i za P2P arhitekturu, uveli smo pojednostavljenje (i u opštem slučaju tačno [Akella 2003]) kroz pretpostavku da u jezgru interneta postoji dovoljno propusnog opsega, da su sva uska grla u pristupnim mrežama. Takođe smo pretpostavili da server i klijenti ne učestvuju ni u jednoj drugoj mrežnoj aplikaciji, tako da se njihov i nizvodni i uzbodni pristupni propusni opseg u potpunosti koristi za distribuiranje ove datoteke.

Prvo određujemo vreme distribucije za klijentsko-serversku arhitekturu, koje označavamo sa D_{cs}. U klijentsko-serverskoj arhitekturi nijedan računar ne pomaže distribuiranje datoteke. Zapažamo sledeće:



Slika 2.24 ◆ Prikaz problema za distribuciju datoteke

- Server mora da prenese jednu kopiju datoteke svakom od N računara. Prema tome, server mora da prenese NF bitova. Pošto je uzvodna brzina servera u_s, vreme distribucije datoteke mora da bude najmanje NF/u_s.
- Neka $d_{\min}$ označava nizvodnu brzinu ravnopravnog računara sa najmanjom brzinom, to jest, $d_{\min} = \min\{d_1, d_2, .., d_N\}$. Računar sa najmanjom nizvodnom brzinom ne može da dobije svih F bitova datoteke brže od $F/d_{\min}$ sekundi. Prema tome, najkraće vreme distribucije je $F/d_{\min}$.

Kada spojimo ova dva zapažanja, dobijamo

$$D_{cs} \geq \max\left\{\frac{NF}{u_s}, \frac{F}{d_{\min}}\right\}.$$

Ovim se dobija donja granična vrednost od koje vreme distribucije za klijentsko-serversku arhitekturu ne može da bude kraće. U domaćem zadatku traži se da pokažete da server može da rasporedi svoj prenos, tako da se ova najmanja vrednost stvarno dostigne. Stoga, za stvarno trajanje distribucije uzimamo ovu donju graničnu vrednost:

$$D_{cs} = \max\left\{\frac{NF}{u_s}, \frac{F}{d_{\min}}\right\}. \quad (2.1)$$

Iz jednačine 2.1 vidimo da je, za dovoljno veliko N, vreme klijentsko-serverske distribucije dato izrazom NF/u_s. Očigledno je da vreme distribucije linearno raste, povećanjem broja ravnopravnih računara N. Recimo, na primer, ukoliko se broj ravnopravnih računara za nedelju dana poveća hiljadu puta, sa hiljadu na milion, vreme potrebno za distribuciju određene datoteke do svih ravnopravnih računara poveća će se 1000 puta.

Sada ćemo sprovesti sličnu analizu za P2P arhitekturu, u kojoj svaki ravnopravni računar može da pomaže serveru u distribuiranju određene datoteke. Drugim rečima, pošto preuzmu deo podataka iz datoteke, računari, koristeći svoje prenosne mogućnosti, dalje distribuiraju te podatke ostalim računarima. Izračunavanje vremena distribucije za P2P arhitekturu je nešto složenije od izračunavanja za klijentsko-serversku arhitekturu, pošto trajanje distribucije zavisi i od toga kako svaki ravnopravni računar distribuira delove datoteke ostalim ravnopravnim računarima. Ipak, može se dobiti jednostavan izraz za najkraće vreme distribucije [Kumar 2006]. Za sada, zapažamo sledeće:

- Na početku distribuiranja, samo server ima datoteku. Da bi se ta datoteka prenela u zajednicu ravnopravnih računara, server mora da svojim pristupnim linkom najmanje jednom pošalje sve bitove. Znači, distribucija ne može da traje kraće od F/u_s. (Za razliku od klijentsko-serverske arhitekture, server ne mora ponovo da šalje bitove koje je jednom već poslao, pošto ravnopravni računari ponovo distribuiraju te bitove između sebe.)
- Kao i u slučaju klijentsko-serverske arhitekture, računar sa najmanjom nizvodnom brzinom ne može da dobije svih F bitova datoteke za manje od sekundi. Znači –minimalno vreme distribucije ne može da bude kraće od F/.
- Konačno, zapažamo da je ukupan uzvodni kapacitet sistema kao celine jednak zbiru uzvodne brzine servera i uzvodnih brzina pojedinih računara, to jest, = + + ...+ . Sistem mora da isporuči (preda) F bitova svakom od N ravnopravnih računara, tako da se isporučuje ukupno NF bitova. To se ne može obaviti brzinom većom od . Znači, distribucija ne može da traje kraće od NF/(+ + ...+).

Spajanjem ovih zapažanja, dobijamo najmanje trajanje distribucije za P2P koje označavamo sa DP2P.

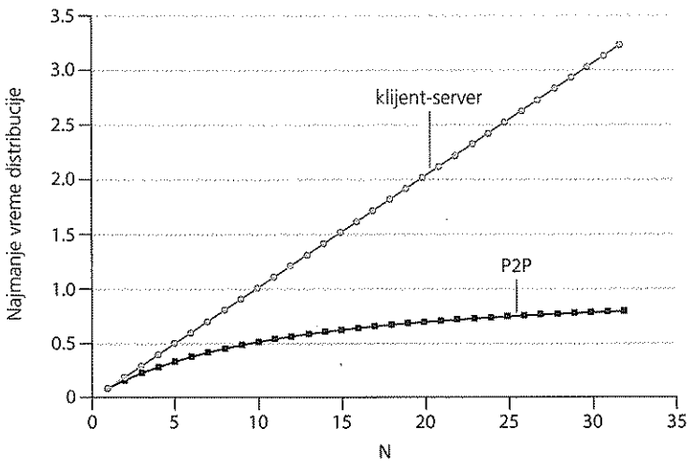
$$D_{\text{P2P}} \geq \max\left\{\frac{F}{u_s}, \frac{F}{d_{\min}}, \frac{NF}{u_s + \sum_{i=1}^{N} u_i}\right\} \quad (2.2)$$

Jednačinom 2.2 se određuje donja granična vrednost minimalnog vremena distribucije za P2P arhitekturu. Iz nje se zaključuje da, ukoliko zamislimo da svi ravnopravni računari primljeni bit ponovo distribuiraju istog trenutka pošto ga preuzmu, onda postoji šema za distribuiranje kojom je zaista moguće dostignuti donju graničnu vrednost [Kumar 2006]. (U domaćim zadacima pokazaćemo poseban slučaj kako se to postiže.) Pošto se u stvarnosti ne prenose pojedinačni bitovi nego delovi datoteke, jednačina 2.2 služi kao dobra aproksimacija

stvarnog minimalnog vremena distribucije. Stoga, za stvarno minimalno vreme distribucije uzimamo donju graničnu vrednost određenu jednačinom 2.2, odnosno

$$D_{\text{P2P}} = \max\left\{\frac{F}{u_s}, \frac{F}{d_{\min}}, \frac{NF}{u_s + \sum_{i=1}^{N} u_i}\right\} \tag{2.3}$$

Na slici 2.25 je prikazano poređenje minimalnog vremena distribucije za klijentsko-serversku i za P2P arhitekturu, pod pretpostavkom da svi ravnopravni računari imaju istu uzvodnu brzinu u. Na slici 2.25 pretpostavili smo da je $F/u = 1$ sat, $u_s = 10u$ i $d_{\min} \geq u_s$. Drugim rečima, ravnopravni računar može da čitavu datoteku prenese za jedan sat, uzvodna brzina prenosa servera je 10 puta veća od uzvodne brzine kojom ravnopravni računar predaje datoteku i pretpostavlja se da je brzina kojom ih preuzimaju dovoljno velika (zbog jednostavnijeg proračuna) tako da nema nikakav uticaj. Sa slike 2.25 vidimo da za klijentsko-serversku arhitekturu trajanje distribucije raste linearno i bez ograničenja sa povećanjem broja ravnopravnih računara. Međutim, za P2P arhitekturu, minimalno vreme distribucije, ne samo da je uvek manje od vremena distribucije za klijentsko-serversku arhitekturu, već je i kraće od jednog sata za *bilo koji* broj ravnopravnih računara N. Ovo znači da se aplikacije u P2P arhitekturi prilagođavaju potrebama. Ova prilagodljivost direktna je posledica toga da ravnopravni računari kao i klijenti učestvuju u preraspodeli bitova.



Slika 2.25 ◈ Vreme distribucije za P2P i klijentsko-serverske arhitekture

BitTorrent

BitTorrent je omiljeni P2P protokol za distribuiranje datoteka [Chao 2011]. U žargonu BitTorrent-a kolekcija svih ravnopravnih računara koji učestvuju u distribuiranju određene datoteke naziva se *torent* (srp. bujica). Ravnopravni računari iz torenta preuzimaju

jednake *odsečke* (eng. *chunk*) datoteke jedan od drugog, pri čemu je uobičajena veličina ovog odsečka 256 Kb. Kada se ravnopravni računar pridruži torentu, on nema nijedan odsečak datoteke koji se distribuira. Tokom vremena na njemu se gomila sve veći broj takvih odsečaka. Istovremeno sa preuzimanjem odsečaka datoteke, računari prenose odsečke datoteke ostalim računarima. Pošto ravnopravni računar preuzme čitavu datoteku, može da (sebično) napusti torent, ili da (velikodušno) ostane u torentu i da nastavi sa prenošenjem odsečaka datoteka ostalim ravnopravnim računarima. Takođe, svaki ravnopravni računar uvek može da napusti torent sa odsečcima datoteka koje je do tada preuzeo i da se kasnije ponovo pridruži torentu.

Pogledajmo sada malo podrobnije kako BitTorrent radi. Pošto je BitTorrent prilično složen protokol, opisujemo samo njegove najvažnije mehanizme, preskačući neke manje važne pojedinosti; tako ćemo videti šumu kroz drveće. Svaki torent ima u infrastrukturi čvor koji se naziva *pratilac* (engl. *tracker*). Kada se ravnopravni računar pridruži torentu, on se prijavljuje pratiocu i povremeno ga obaveštava o tome da je još uvek deo torenta. Na taj način, pratilac vodi računa o računarima koji učestvuju u torentu. U određenim torentima povremeno mogu da učestvuju stotine ili hiljade ravnopravnih računara.

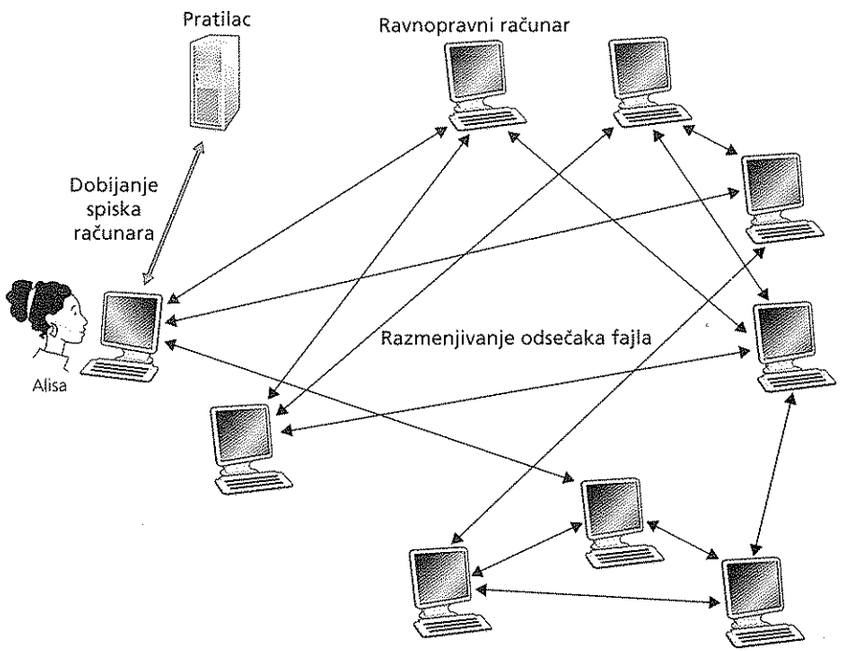
Kao što je prikazano na slici 2.26, kada se novi ravnopravni računar, Alisa, pridruži torentu, pratilac nasumice bira određeni broj drugih ravnopravnih računara (recimo 50) iz skupa ravnopravnih računara koji učestvuju u tom torentu i Alisi šalje IP adrese tih 50 računara. Dobivši taj spisak računara, Alisa pokušava da uspostavi istovremene TCP veze sa svim računarima sa spiska. Sve računare sa kojima Alisa uspešno uspostavi TCP vezu nazvaćemo „susedni računari". (Na slici 2.26 je prikazano da Alisa ima samo tri susedna računara. Obično ih ima mnogo više.) Vremenom neki od tih ravnopravnih računara odlaze, a neki drugi (osim početnih 50) pokušavaju da sa Alisom uspostave TCP vezu. Stoga se tokom vremena broj susednih računara stalno menja.

U nekom trenutku, svaki ravnopravni računar će imati po nekoliko odsečaka određene datoteke, pri čemu različiti ravnopravni računari imaju različite odsečke. Povremeno, Alisa od svojih susednih računara traži (preko TCP veze) spisak odsečaka koji oni poseduju. Ukoliko Alisa ima L različitih suseda, dobiće L spiskova odsečaka datoteke. Kada to sazna, Alisa šalje zahtev (ponovo preko TCP veze) za one odsečke koje trenutno nema.

I tako, u nekom trenutku vremena, Alisa ima nekoliko odsečaka datoteka i zna koje odsečke imaju njeni susedi. Uz pomoć te informacije, Alisa mora da donese dve važne odluke. Pre svega, koje odsečke prvo da zatraži od svojih suseda? I drugo, kojim svojim susedima da pošalje tražene odsečke? Pri odlučivanju o tome koje odsečke da zatraži, Alisa koristi tehniku koja se naziva **prvo najređi**. Ideja je u tome da odredi, među odsečcima koje nema, koje odsečke njeni susedi najređe imaju (to jest, odsečke čije se kopije najmanje puta ponavljaju među njenim susedima), a da zatim prvo zatraži te najređe odsečke. Na taj način, najređi odsečci se mnogo brže prenose dalje, čime se (bar približno) izjednačava broj kopija svih odsečaka u torentu.

Da bi odredio na koje zahteve Alisa odgovara, BitTorrent koristi pametan algoritam za razmenu. Osnovna ideja je da Alisa daje prednost onim susedima koji joj trenutno isporučuju podatke *najvećom brzinom*. Drugim rečima, za sve svoje susede Alisa neprestano meri brzinu kojom od njih prima bitove i određuje četiri ravnopravna računara koji

joj bitove isporučuju najvećom brzinom. Na osnovu toga ona odgovara slanjem svojih odsečaka na ta četiri računara. Svakih 10 sekundi, ona ponovo preračunava brzine i po potrebi menja skup ova četiri ravnopravna računara. U žargonu BitTorrenta, za ova četiri ravnopravna računara se kaže da su **neprigušeni**. Što je još važnije, svakih 30 sekundi, nasumice bira jednog od svojih suseda i šalje mu odsečke. Nazovimo tog slučajno izabranog suseda Bob. U žargonu BitTorenta, za Boba se kaže da je **optimistično neprigušen**. Pošto Alisa šalje Bobu podatke, ona može da postane jedna od četiri najbrža Bobova dostavljača; u tom slučaju i Bob počinje Alisi da dostavlja podatke. Ukoliko je brzina kojom Bob dostavlja podatke Alisi dovoljno velika, Bob tako može, sa svoje strane, da postane jedan od četiri najbrža Alisina dostavljača. Drugim rečima, svakih 30 sekundi, Alisa nasumice bira novog partnera za razmenu i započinje razmenu sa njim. Ukoliko su dva ravnopravna računara zadovoljna razmenom, oni jedan drugog stavljaju na spisak četiri najbolja dobavljača i nastavljaju međusobnu razmenu sve dok jedan od njih ne pronađe boljeg partnera. Posledica je težnja ravnopravnih računara da pronađu druge računare, koji mogu da razmenjuju podatke brzinama približnim njihovim brzinama. Nasumično biranje suseda takođe omogućava novopristiglim, ravnopravnim računarima da dobiju odsečke, tako da imaju šta da ponude za razmenu. Svi ostali susedni ravnopravni računari, osim ovih pet (četiri „najbolja" računara i jedan osnovni) su „prigušeni", odnosno od Alise ne dobijaju odsečke. BitTorrent ima brojne druge zanimljive mehanizme o kojima ovde nismo govorili, kao što su oni za slanje manjih delova (mini-odsečaka), preklapanje tokova, nasumičan prvi izbor, prekid rada i ravnopravnost korisnika [Cohen 2003].



Slika 2.26 ◆ Distribucija datoteka protokolom BitTorrent

Podsticajni mehanizmi za razmenu, opisani gore, često se zovu tit-for-tat algoritmi [Cohen 2003]. Prikazano je da se ova podsticajna šema može osujetiti [Liogkas 2006; Locher 2006; Piatek 2007]. Pa, ipak BitTorrent eko sistem je neverovatno uspešan, sa milionima simultanih ravnopravnih računara koji aktivno dele datoteke u stotinama hiljada torenata. Da BitTorrent nije dizajniran sa algoritmom tit-for-tat (ili varijantom), ali s druge strane potpuno isti, BitTorrent verovatno sada ne bi postojao, jer bi većina korisnika bila slobodni strelci [Saroiu 2002].

Zanimljivu varijantu protokola BitTorrent predložili su [Guo 2205; Piatek 2007]. Takođe, mnoge P2P aplikacije koje reprodukuju tokove podataka, kao što je PPLive i ppstream, inspirirao je BitTorrent [Hei 2007].

2.6.2 Distribuirane heš tabele

U ovom odeljku razmotrićemo implementiranje jednostavne baze podataka u P2P mreži. Počećemo tako što ćemo opisati centralizovanu verziju jednostavne baze podataka, koja će jednostavno sadržati (ključ, vrednosti) parove. Na primer, ključevi mogu da budu brojevi socijalnog osiguranja, a vrednosti mogu biti imena ljudi; u ovom slučaju, primer je jednog para ključ – vrednost je [156-45-7081, Johnny Wu]. Ključevi mogu biti i nazivi sadržaja (npr. nazivi filmova, albuma i softvera), a vrednost može da bude IP adresa na kojoj je sadržaj sačuvan; u ovom slučaju, ovo je jedan primer para ključ – vrednost (Led Zeppelin IV, 128.17.123.38). Bazi podataka šaljemo upit pomoću ključa. Ukoliko postoji jedan ili više parova ključ – vrednost u bazi podataka, koji odgovaraju zadatom upitu preko ključa, baza podataka će dati odgovarajuće vrednosti. Tako, na primer, ako baza podataka čuva brojeve socijalnog osiguranja i odgovarajuća imena ljudi, možemo izvršiti upit na osnovu određenog broja socijalnog osiguranja i baza podataka će dati ime čoveka sa tim brojem socijalnog osiguranja. Baza podataka može da sadrži i nazive sadržaja i njihove odgovarajuće IP adrese, pa možemo izvršiti upit pomoću određenog naziva sadržaja, a baza podataka će vratiti IP adrese koje čuvaju određeni sadržaj.

Gradnja ovakve baze podataka je jednostavna pomoću klijentsko-serverske arhitekture koja skladišti sve parove (ključ, vrednost) na jednom centralnom serveru. Stoga ćemo se u ovom odeljku baviti gradnjom distribuirane, P2P verzije ove baze podataka koja će skladištiti parove (ključ, vrednost) miliona ravnopravnih računara. U P2P sistemu, svaki ravnopravni računar će sadržati samo mali podskup svih parova (ključ, vrednost). Dozvolićemo svakom ravnopravnom računaru da pošalje upit distribuiranoj bazi podataka pomoću određenog ključa. Distribuirana baza podataka će zatim pronaći ravnopravne računare koji imaju odgovarajuće parove (ključ, vrednost) i dati parove ključ-vrednost ravnopravnom računaru koji je poslao upit. Svakom ravnopravnom računaru će biti dozvoljeno da u bazu podataka unese nove parove ključ-vrednost. Takva distribuirana baza podataka se naziva **distribuirana heš tabela (DHT)**.

Video napomena Kretanje kroz distribuirane heš tabele

Pre nego što opišemo kako možete da kreirate distribuiranu heš tabelu, prvo ćemo objasniti karakterističan primer DHT usluge u kontekstu deljenja P2P datote-

ke. U ovom slučaju, ključ je naziv sadržaja, a vrednost je IP adresa ravnopravnog računara, na kome se nalazi kopija sadržaja. Stoga, ako i Bob i Čarli imaju kopiju najnovije Linux distribucije, onda će DHT baza podataka sadržati sledeća dva para ključ-vrednost: (Linux, IP_{Bob}) i (Linux, $IP_{Charlie}$). Tačnije, s obzirom da je DHT baza podataka distribuirana preko ravnopravnih računara, neki računar, recimo Dejv, biće odgovoran za ključ „Linux" i imaće odgovarajuće parove ključ-vrednost. Sada pretpostavimo da Alisa želi da nabavi kopiju distribucije Linux. Jasno je da Alisa mora da zna koji ravnopravni računari imaju kopiju distribucije Linux, pre nego što počne da ga preuzima. Do sada je DHT ispitivala pomoću ključa „Linux". DHT zatim određuje da je ravnopravni računar Dejv odgovoran za ključ „Linux". DHT tada kontaktira ravnopravni računar Dejv, dobija od Dejva parove ključ-vrednost (Linux, IP_{Bob}) i (Linux, $IP_{Charlie}$) i prosleđuje ih do Alise. Alisa tada može da preuzme najnoviju Linux distribuciju od IP_{Bob} ili $IP_{Charlie}$.

Vratimo se sada opštem problemu dizajniranja distribuirane heš tabele za opšte parove ključ-vrednost. Jedan naivan pristup građenja DHT-a je nasumično grupisanje parova (ključ, vrednost) duž svih ravnopravnih računara, pri čemu će svaki ravnopravni računar održavati listu IP adresa svih ravnopravnih računara koji učestvuju. U ovom dizajnu, ravnopravni računar koji izvršava upit šalje svoj upit ostalim ravnopravnim računarima, a računari koji sadrže parove (ključ, vrednost), koji se podudaraju sa ključem mogu da se odazovu sa svojim uparenim parovima. Ovakav pristup je, naravno, potpuno neprilagodljiv jer zahteva da svaki ravnopravni računar zna sve o svim drugim ravnopravnim računarima (verovatno milionima ovakvih računara) i , što je čak i gore, da svaki upit šalje *svim* ovim računarima.

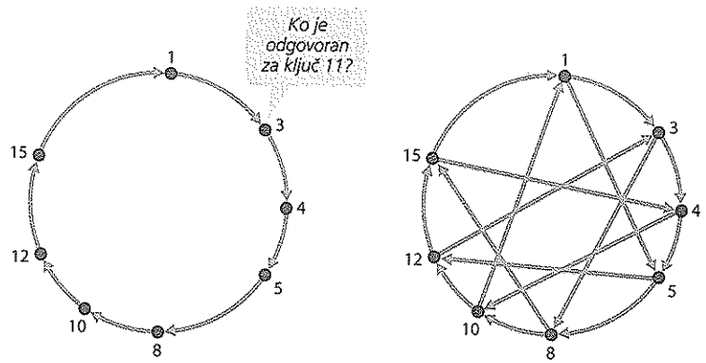
Objasnićemo sada jedan elegantan pristup dizajniranja distribuirane heš tabele. Stoga, dodelićemo prvo svakom ravnopravnom računaru identifikator, koji je predstavljen celim brojem u opsegu $[0, 2^n-1]$ za neko stalno n. Primetite da svaki takav identifikator može da se iskaže kao n-bitni prikaz. Takođe, zahtevaćemo da svaki ključ bude ceo broj u istom opsegu. Lukavi čitalac može da primeti da primer ključeva, opisanih malo ranije, (broj socijalnog osiguranja i nazivi sadržaja) nisu celi brojevi. Da bi se kreirali celi brojevi za ove ključeve, koristićemo heš funkciju koja preslikava svaki ključ (npr. broj socijalnog osiguranja) u ceo broj u opsegu $[0, 2^n-1]$. Heš funkcija je jedna od mnogih funkcija za koje dva različita ulaza mogu dati isti rezultat (isti ceo broj), ali je verovatnoća da se dobije isti rezultat ekstremno mala. (Čitaoci koji nisu upoznati sa heš funkcijama trebalo bi da pogledaju poglavlje 7, u kome su heš funkcije detaljno objašnjene). Pretpostavlja se da je heš funkcija raspoloživa za sve ravnopravne računare u sistemu. Od sada, kada se pozovemo na „ključ", mislimo na heš originalnog ključa. Na primer, ako je originalni ključ „Led Zeppelin IV", ključ koji se koristi u distribuiranoj heš tabeli biće ceo broj koji je jednak hešu „Led Zeppelin IV". Kao što ste možda pretpostavili, zbog toga se „Heš" koristi u nazivu „Distribuirana heš funkcija".

Sada ćemo se pozabaviti problemom skladištenja parova (ključ, vrednost) u distriburiranoj heš tabeli. Ovde je centralno pitanje definisanje pravila za dodeljivanje ključeva ravnopravnim računarima. S obzirom da svaki ravnopravni računar ima identifikator u obliku celog broja i da je svaki ključ takođe ceo broj u istom opsegu, prirodni pristup bio bi dodeljivanje svakog para (ključ, vrednost) ravnopravnom računaru čiji je identifikator *najbliži* ključu. Da bi se ovakva šema primenila, potrebno je da se definiše šta znači „najbliži" za koje su mnoge konvencije moguće. Da bismo pojednostavili, definisaćemo najbliži ravnopravni računar kao *najbližeg sledbenika ključa*. Da bi stekli neki uvid o ovome, pogledajmo karakterističan primer. Pretpostavimo da je $n = 4$, tako da su svi identifikatori ravnopravnog računara i ključa u istom opsegu [0, 15]. Pretpostavimo potom da postoji osam ravnopravnih računara u sistemu čiji su identifikatori: 1, 3, 4, 5, 8, 10, 12 i 15. Konačno, pretpostavimo da želimo da sačuvamo par (ključ, vrednost) (11, Džoni Vu) na jednom od osam ravnopravnih računara. Ali kojem? Koristeći našu konvenciju najbližeg, pošto je 12 najbliži sledbenik ključa 11, mi ćemo skladištiti par (11, Džoni Vu) na računaru 12. [Da bi kompletirali našu definiciju najbližeg, ukoliko je ključ jednak jednom od identifikatora ravnopravnih računara, par (ključ, vrednost) ćemo sačuvati na tom ravnopravnom računaru; a ako je ključ veći od svih identifikatora ravnopravnih računara, koristićemo konvenciju moduo , gde ćemo par (ključ, vrednost) sačuvati na ravnopravnom računaru sa najmanjim identifikatorom.]

Pretpostavimo sada da ravnopravni računar Alisa, želi da unese par (ključ, vrednost) u distribuiranu heš tabelu. Teorijski, ovo je jednostavno: ona prvo mora da odredi koji ravnopravni računar ima identifikator koji je najbliži ključu; tada šalje poruku tom ravnopravnom računaru, dajući mu instrukcije da sačuva par (ključ, vrednost). Ali, kako će Alisa utvrditi koji je ravnopravni računar najbliži ključu? Ako bi Alisa pratila sve ravnopravne računare u sistemu (identifikatore ravnopravnih računara i odgovarajuće IP adrese), mogla bi lokalno da utvrdi najbliži ravnopravni računar. Međutim, ovakav pristup zahteva da *svaki* ravnopravni računar prati *sve* ravnopravne računare u distribuiranoj heš tabeli – što je potpuno nepraktično za velike sisteme sa milionima ravnopravnih računara.

Cirkularna distribuirana heš tabela

Kako bismo se pozabavili ovim problemom vezanim za veličinu, uzećemo sada u obzir organizovanje ravnopravnih računara u krug. U ovom kružnom rasporedu, svaki ravnopravni računar prati samo računar koji mu neposredno sledi i prethodi (po modulu). Primer ovakvog kruga prikazan je na slici 2.27 (a). U ovom primeru, n je ponovo 4 i postoji istih osam ravnopravnih računara kao uprethodnom primeru.



Slika 2.27 ◆ (a) Cirkularni DHT. Ravnopravni računar 3 želi da utvrdi ko je odgovoran za ključ 11. (b) Cirkularni DHT sa prečicama

Svaki ravnopravni računar prati samo o svog neposrednog sledbenika i prethodnika; na primer, ravnopravni računar 5 zna IP adresu i identifikator računara 8 i 4, ali nije neophodno da zna bilo šta o bilo kom drugom ravnopravnom računaru koji može da se nalazi u distribuiranoj heš tabeli. Kružni raspored ravnopravnih računara je specifičan slučaj **preklopljene mreže**. U preklopljenoj mreži ravnopravni računari formiraju apstraktnu logičku mrežu koja se nalazi iznad računarske mreže koja sadrži fizičke linkove, rutere i računare hostove. Linkovi u preklopljenoj mreži nisu fizički linkovi, već jednostavno virtuelne veze parova ravnopravnih računara. U svakom preklapanju na slici 2.27 (a) postoji osam ravnopravnih računara i osam preklopljeni linkova; u preklapanju na slici 2.27 (b) postoji osam ravnopravnih računara i 16 preklopljeni linkova. Pojedinačan preklopljeni link obično koristi brojne fizičke linkove i fizičke rutere u stvarnoj fizičkoj mreži.

Koristeći kružno preklapanje sa slike 2.27 (a), pretpostavimo sada da ravnopravni računar 3 želi da odredi koji ravnopravni računar u DHT je odgovoran za ključ 11. Pomoću kružnog preklapanja, izvorni ravnopravni računar (računar 3) kreira poruku koja glasi: „Ko je odgovoran za ključ 11?" i ovu poruku šalje u pravcu kazaljke na satu duž kruga. Svaki put kada ravnopravni računar primi ovakvu poruku, a pošto zna identifikator svog sledbenika i prethodnika, on može da odredi da li je on odgovoran za ključ (koji je najbliži) iz pitanja. Ako ravnopravni računar nije odgovoran za ključ, on jednostavno šalje poruku svom sledbeniku. Tako, na primer, kada ravnopravni računar 4 primi poruku koja ga pita za ključ 11, on određuje da nije odgovoran za ključ (jer je njegov sledbenik bliži ključu), i prosleđuje poruku do ravnopravnog računara 5. Ovaj proces se nastavlja dok poruka ne stigne do ravnopravnog računara 12, koji utvrđuje da je najbliži ravnopravni računar ključu 11. Na ovom mestu, ravnopravni računar 12 može da pošalje poruku nazad do ravnopravnog računara koji je poslao upit, računara 3, ukazujući da je on odgovoran za ključ 11.

Kružna distribuirana heš tabela pruža veoma elegantno rešenje za smanjenje količine preklopljenih informacija kojima svaki ravnopravni računar mora da upravlja. Naročito, svaki ravnopravni računar trebalo bi da prati samo dva druga ravnopravna računara, svog neposrednog sledbenika i prethodnika. Međutim, ovo rešenje nas upoznaje sa novim problemom. Iako svaki ravnopravni računar prati samo dva susedna računara, da bi se pronašao čvor odgovoran za ključ (u najgorem slučaju), svih N čvorova u DHT moraće da prosledi poruku duž kruga; u proseku se šalje $N/2$ poruke.

Stoga, u dizajniranju distribuirane heš tabele, postoji kompromis između broja suseda koje bi svaki ravnopravni računar trebalo da prati i broja poruka koje bi DHT trebalo da pošalje kako bi se rešio pojedinačni upit. S jedne strane, ukoliko svaki ravnopravni računar prati sve druge ravnopravne računare (mrežasto preklapanje), tada će se slati jedna poruka po upitu, ali će svaki računar morati da prati N ravnopravnih računara. S druge strane, pomuću kružne distribuirane heš tabele, svaki ravnopravni računar će biti svestan samo dva ravnopravna računara, ali će se prosečno po svakom upitu slati $N/2$ poruke. Srećom, možemo da doteramo naš dizajn distribuiranih heš tabela tako da broj suseda po računaru kao i broj poruka po upitu bude prihvatljive veličine. Jedno takvo doterivanje je korišćenje kružnog preklapanja kao osnove, ali i dodavanje „prečica", tako da svaki ravnopravni računar prati svog neposrednog sledbenika i prethodnika, kao i relativno mali broj ravnopravnih računara preko prečica koji su grupisani oko kruga. Primer ovakve kružne distribuirane heš tabele sa nekim prečicama prikazan je na slici 2.27 (b). Prečice se koriste za ekspresno usmeravanje poruka upita. Posebno, kada ravnopravni računar primi poruku koja šalje upit vezan za ključ, on prosleđuje poruku svom susedu (sused koji sledi ili jedan od suseda na prečici) koji je najbliži ključu. Stoga, na slici 2.27 (b) kada ravnopravni računar 4 primi poruku koja ga pita o ključu 11, on određuje da je najbliži ravnopravni računar ključu (među susedima) njegov sused na prečici 10 i tada prosleđuje poruku direktno računaru 10. Jasno je da prečice značajno mogu da smanje broj poruka koje se koriste prilikom obrade upita.

Sledeće prirodno pitanje je: „Koliko bi suseda na prečici ravnopravni računar trebalo da ima i koji bi ravnopravni računari trebalo da budu susedi na prečici?" Ovo pitanje je izazavalo veliku pažnju u istraživačkoj zajednici [Balakrishnan 2003; Androutsellis Thetokis 2004]. Važno je da je prikazano da distriburiane heš tabele mogu biti dizajnirane tako da i broj suseda po ravnopravnom računaru i brojevi poruka po upitu budu srazmerni O (log N), gde je N broj ravnopravnih računara. Takav dizajn dovodi do zadovoljavajućeg kompromisa između ekstremnih rešenja, korišćenje mrežastih i kružnih topoligija preklapanja.

Oscilacije ravnopravnih računara

U P2P sistemima ravnopravni računar može da se pojavi i nestane bez upozorenja. Stoga, prilikom dizajniranja distribuirane heš tabele, moramo uzeti u obzir održavanje DHT preklapanja u slučaju ovih oscilacija ravnopravnih računara. Da bi malo bolje razumeli kako ovo može da se postigne, još jednom pogledajmo kružni DHT na slici 2.27 (a). Da bi upravljali oscilacijama ravnopravnih računara, potrebno je da svaki ravnopravni računar prati (odnosno zna IP adresu) svog prvog i drugog sledbenika; na primer, računar 4 prati računare 5 i 8. Takođe, potrebno je da svaki ravnopravni računar periodično proverava da li su njegova dva sledbenika živa (na primer, periodičnim slanjem ping poruka i zahtevanjem odgovora od njih). Pogledajmo sada kako se održava distribuirana heš tabela kada ravnopravni računar iznenada nestane. Na primer, pretpostavimo da računar 5 na slici 2.27 (a) iznenada ode. U ovom slučaju, dva ravnopravna računara koja prethode računaru koji je otišao (4 i 3) će shvatiti da je 5 otišao, jer više ne odgovara na ping poruku. Sada bi računari 4 i 3 trebalo da ažuriraju stanje inforamicija njihovog sledbenika. Pogledajmo sada kako računar 4 ažurira svoje stanje:

1. Ravnopravni računar 4 menja svog prvog sledbenika (računar 5) svojim drugim sledbenikom (računar 8).
2. Ravnopravni računar 4 traži od novog prvog sledbenika (računar 8) identifikator i IP adresu njegovog neposrednog sledbenika (računar 10). Računar 4 tada postavlja za svog drugog sledbenika računar 10.

U problemima u domaćim zadacima, tražiće se od vas da odredite kako računar 3 ažurira svoje preklopljene informacije usmeravanja.

Pošto smo se ukratko pozabavili time šta bi trebalo da se uradi kada ravnopravni računar ode, pogledajmo šta se dešava kada ravnopravni računar želi da se

priključi distribuiranoj heš tabeli. Recimo da računar sa identifikatorom 13 želi da se priključi distribuiranoj heš tabeli, i da u vreme pristupanja, on zna samo za postojanje računara 1 u DHT-u. Računar 13 će prvo poslati poruku računaru 1, u kojoj će reći: „ko će biti prethodnik i sledbenik računara 13?" Ova poruka se prosleđuje kroz distribuiranu heš tabelu sve dok ne stigne do računara 12, koji shvata da će biti prethodnik računara 13 i da je on njegov trenutni sledbenik, a da će računar 15 postati sledbenik računara 13. Zatim, računar 12 šalje informacije o prethodniku i sledbeniku računaru 13. Ravnopravni računar 13 može sada da se priključi distribuiranoj heš tabeli, čineći da računar 15 bude njegov sledbenik uz obaveštavanje računara 12 da bi trebalo da zameni svog neposrednog sledbenika u računar 13.

Distribuirane heš tabele široko su rasprostranjene u praksi. Na primer, BitTorrent koristi Kademlia DHT za kreiranje distribuiranog tragača. U BitTorrentu ključ je identifikator torenta, a vrednost su IP adrese svih ravnopravnih računara koji trenutno učestvuju u porotoku torent podataka [Falkner 2007, Neglia 2007]. Na ovaj način, šaljući upit koji sadrži torent identifikator, novopridošli ravnopravni računar za BitTorrent može da odredi ravnopravni računar koji je odgovoran za identifikator (odnosno, za praćenje računara u torentu). Nakon što pronađe ravnopravni računar, pridošli računar može da pošalje upit sa listom drugih ravnopravnih računara u torentu.

2.7 Programiranje soketa: Kreiranje mrežnih aplikacija

Pošto smo pregledali nekoliko važnijih mrežnih aplikacija, pogledajmo kako se ovakvi mrežni programi zaista i pišu. Ako se sećate, u odeljku 2.1 smo rekli da se većina mrežnih aplikacija sastoji od dva programa – klijentskog i serverskog programa – koji se nalaze na dva različita krajnja sistema. Izvršavanjem ova dva programa kreiraju se klijentski i serverski procesi koji između sebe komuniciraju čitanjem iz soketa i pisanjem u njih. Osnovni zadatak programera prilikom kreiranjamrežne aplikacije jeste pisanje koda za klijentski i serverski program.

Postoje dve vrste mrežnih aplikacija. Prvu vrstu čine implementacije čiji je rad definisan u standardima protokola, kao što su RFC ili neka druga dokumenta sa standardima; ovakve aplikacije se nekad zovu „otvorenim", jer su pravila kojima je definisan njihov rad poznate svima. U tu svrhu, kod takve imlementacije klijentski i serverski program moraju da budu u skladu sa pravilima koja postavljaju RFC dokumenti. Na primer, klijentski program bi mogao da predstavlja implementacij u klijentske strane protokola FTP opisanog u odeljku 2.3 i koji je eksplicitno definisan u dokumentu RFC 959; slično tome, serverski program mogao bi da bude implementacija serverske strane protokola FTP, koji je takođe definisana u dokumentu RFC 959. Ukoliko jedan programer napiše kôd za klijentski program, a neki drugi, za serverski i pri tom se oba dosledno pridržavaju pravila postavljenih u RFC dokumentima, onda će ta dva programa moći međusobno da sarađuju. U stvari, u velikom broju savremene mrežne aplikacije podrazumevaju komunikaciju između klijentskih i serverskih programa koje su potpuno nezavisno napravili različiti programeri – na primer, veb pretraživač *Firefox* koji komunicira sa *Apache veb* serverom ili BitTorrent klijent koji komunicira sa BitTorrent tragačem.

Drugu vrstu mrežnih aplikacija čine vlasničke mrežne aplikacije. U ovom slučaju, protokol aplikativnog sloja koji koriste klijentski i serverski programi *nije* javno objavljen u nekom RFC dokumentu ili negde drugde. Jedan programer (ili tim programera) prave i klijentski i serverski program i samo ti programeri imaju potpuni uvid u to što se dešava u kodu. Ali, budući da kôd ne implementira javno dostupne protokole, ostali nezavisni programeri ne mogu da razvijaju aplikacije koje bi sarađivale sa tim programima.

U ovom odeljku ispitaćemo ključna pitanja u razvoju klijentsko-serverskih aplikacija, a pozabavićemo se i kodom koji implementira veoma jednostavnu klikentsko-serversku aplikaciju. Tokom faze razvoja, jedna od prvih odluka koju programeri moraju da donesu, jeste da li će se aplikacija izvršavati pomoću protokola TCP ili protokola UDP. Podsećamo vas da je TCP protokol sa uspostavljenjem veze i da obezbeđuje pouzdane kanale za tokove bajtova između dva krajnja sistema. Protokol UDP ne uspostavlja vezu i njime se nezavisni paketi podataka šalju od jednog do drugog krajnjeg sistema, bez ikakvih garancija da će biti isporučeni. Setite se takođe da, kada klijentski ili serverski program implementira protokol definisan dokumentom RFC, on bi trebalo da koristi dobro poznati broj porta povezan sa protokolom; obrnuto, kada se razvija vlasnička aplikacija, programer mora pažljivo da izbegne korišćenje dobro poznatih brojeva portova. (Brojevi portova su ukratko objašnjeni u odeljku 2.1. Detaljnije su obrađeni u poglavlju 3.)

Predstavićemo UDP i TCP programiranje soketa kroz jednostavne UDP i TCP aplikacije. Ove jednostavne TCP i UDP aplikacije predstavljene su u programskom jeziku *Python*. Programe smo mogli da napišemo i u programskim jezicima *Java*, *C* ili *C++*, ali smo se opredelili za *Python*, pošto su ključni koncepti soketa mnogo jasnije predstavljeni u ovom programskom jeziku. U programskom jeziku *Python* postoji manje redova koda, pri čemu se svi mogu objasniti programeru početniku bez ikakvih teškoća. Nema razloga da se plašite ukoliko ne poznajete *Python*. Programersko iskustvo u jezicima *Java*, *C* ili *C++* pomoći će vam da ovaj kôd pratite bez ikakvih problema.

Ukoliko ste zainteresovani za klijentsko-serversko programiraje u programu *Java*, preporučujemo vam da pogledate prateću veb lokaciju ove knjige; u stvari, tamo ćete naći sve primere iz ovog odeljka (i pridružene laboratorije) u programskom jeziku *Java*. Za čitaoce koje posebno zanima programiranje klijentsko-serverskih aplikacija u programskom jeziku *C* postoji nekoliko dobrih knjiga na tu temu [Donahoo 2001; Stevens 1997; Frost 1994; Kurose 1996]; naši primeri ispod u programu *Python* slično izgledaju i ponašaju se kao programski jezik *C*.

2.7.1 Programiranje soketa za protokol UDP

U ovom odeljku pišemo jednostavne klijentsko-serverske programe koji koriste protokol UDP; u sledećem odeljku pišemo slične programe koji koriste prokol TCP.

Setite se iz odeljka 2.1 da procesi koji se izvršavaju na različitim uređajima komuniciraju između sebe slanjem poruka u soket. Kažemo da je svaki proces analogan sa kućom, a soket procesa je analogan sa vratima. Aplikacija se nalazi sa unutrašnje strane vrata; protokol transportnog sloja se nalazi na drugoj strani vrata u

spoljnom svetu. Programer aplikacije ima kontrolu nad svim na strani aplikativnog sloja soketa; međutim, ima malu kontrolu sa strane transportnog sloja.

Pogledajmo sada detaljnije interakciju između dva komunikaciona procesa koji koriste sokete protokola UDP. Pre nego što proces koji šalje može da preda paket podataka kroz vrata soketa, prilikom upotrebe protokola UDP, on mora prvo da pripoji odredišnu adresu paketu. Pošto se paket prosledi kroz soket pošiljaoca, internet će koristiti ovu odredišnu adresu za usmeravanje paketa kroz internet ka soketu prijemnog procesa. Kada paket stigne u prijemni soket, prijemni proces će preuzeti paket kroz soket, i zatim će prekontrolisati sadržaj paketa i preduzeti akciju.

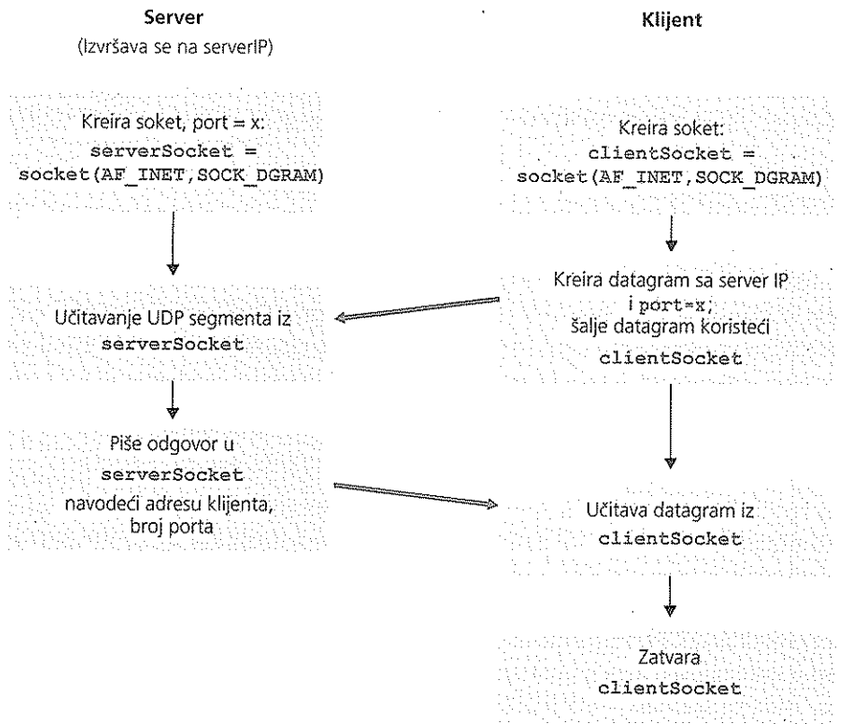
Možda ćete se sada zapitati, šta se dešava na odredišnoj adresi koja je pripojena paketu? Kao što možete da očekujete, IP adresa odredišnog računara je deo odredišne adrese. Uključivanjem odredišne IP adrese paketu, ruteri na internetu će moći da usmere paket kroz internet do odredišnog računara. Međutim, pošto računar izvršava brojne procese mrežnih aplikacija, svaki sa jednim ili više soketa, neophodno je identifikovati pojedinačan soket na odredišnom računaru. Kada se kreira soket, dodeljuje mu se identifikator koji se naziva **broj porta**. Tako da, kao što možete i da pretpostavite, odredišna adresa paketa uključuje takođe i broj porta soketa. Uopšteno, proces slanja pripaja paketu odredišnu adresu koja sadrži IP adresu odredišnog računara i broj porta odredišnog soketa. Šta više, kao što ćete uskoro videti, izvorna adresa pošiljaoca – koja se sastoji iz IP adrese izvornog računara i broja porta izvornog soketa – takođe se pripaja paketu. Međutim, pripajanje izvorne adrese paketu tipično se *ne* radi pomoću koda UDP aplikacije ; to se radi automatski pomoću operativnog sistema u upotrebi.

Koristićemo sledeću jednostavnu klijentsko-serversku aplikaciju da bi pokazali programiranje soketa za protokole UDP i TCP:

1. Klijent učitava red karaktera (podataka) sa tastature i šalje podatke serveru.
2. Server prima podatke i konvertuje karaktere u velika slova.
3. Server šalje izmenjene podatke klijentu.
4. Klijent preuzima izmenjene podatke i prikazuje red na svom ekranu.

Slika 2.28 prikazuje glavne aktivnosti vezane za sokete klijenta i servera koji komuniciraju preko transportne usluge protokola UDP.

Pozabavimo se sada i pogledajmo klijentskii serverski program za UDP implementaciju ove jednostavne aplikacije.Takođe pokazaćemo detaljnu, red po red analizu svakog programa. Počećemo od UDP klijenta,koji šalje jednostavnu poruku aplikativnog niova serveru. Kako bi server mogao da preuzme i odgovori na poruku klijenta, mora da bude spreman i da je već pokrenut – to znači, mora da se izvršava kao proces, pre nego što klijent pošalje svoju poruku.



Slika 2.28 ◆ Klijentsko-serverska aplikacija pomoću protokola UDP

Klijentski program se naziva UDPClient.py, a serverski UDPServer.py. Kako bi naglasili ključna pitanja, namerno ćemo obezbediti minimalni kôd. „Dobar kôd" će imati malo više pomoćnih redova, posebno za slučajeve upravljanja greškama. Za ovu aplikaciju smo slučajno izabrali 12 000, da bude broj porta servera.

Program UDPClient.py

Evo kako izgleda kôd klijentske strane aplikacije:

```
fromsocketimport*
serverName='hostname'serverPort=12000
clientSocket=socket(socket.AF_INET,socket.SOCK_DGRAM)
message=raw_input('Input lowerca sesentence:')
clientSocket.sendto(message,(serverName,serverPort))
modifiedMessage,serverAddress=clientSocket.recvfrom(2048)
printmodifiedMessage
clientSocket.close()
```

Pogledajmo sada različite redove koda za klijentski program UDPClient.py.

```
fromsocketimport*
```

Modul `socket` formira osnovu svih mrežnih komunikacija u programskom jeziku *Python*.Uključujući ovaj red, moći ćemo da kreiramo sokete unutar našeg programa.

```
serverName ='hostname'
serverPort = 12000
```

Prvi red dodeljuje vrednost string-u server Name. Ovde ćemo obezbediti da string sadrži IP adresu servera (npr. „128.138.32.126"), ili ime računara servera (npr. „cis.poly.edu"). Ako koristimo naziv računara, onda će se automatski izvršiti DNS pretraživanje, kako bi se dobila IP adresa). Drugi red dodeljuje vrednost celobrojnoj promenljivi server Port na 12000.

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

Ovaj red kreira soket klijenta, koji se naziva clientSocket. Prvi parametar ukazuje na familiju adrese; naime, AF_INET ukazuje da navedena mreža koristi IPv4. (Ne brinite sada o ovome – o IPv4 ćemo govoriti u poglavlju 4.) Drugi parametar ukazuje da je tip soketa SOCK_DGRAM, što znači da je to soket protokola UDP (a ne soket protokola TCP). Primetite da ne navodimo broj porta klijentskog soketa kada ga keriramo; umesto toga prepuštamo operativnom sistemu da uradi to umesto nas. Sada, kada su kreirana vrata procesa klijenta, želećemo da kreiramo poruku koju ćemo poslati kroz ta vrata.

```
message=raw_input('Inputlowercasesentence:')
```

raw_input je funkcija ugrađena u programskom jeziku *Python*. Kada se ova komanda izvršava, od korisnika klijenta se sledećim rečima zahteva da se odazove „Input data: (Unesi podatke)". Korisnik tada koristi svoju tastaturu da unese red, koji se stavlja u promenljivu message. Sada, kada imamo soket i poruku, želimo da pošaljemo poruku kroz soket do odredišnog računara.

```
clientSocket.sendto(message,(serverName,serverPort))
```

Metoda u gornjem redu: „sendto()"pripaja odredišnu adresu (server Name, server Port)porucii šalje, rezultujući paket u soket procesa clientSocket. (Kao što smo ranije napomenuli, izvorna adresa je takođe pripojena paketu, iako se ovo pre vrši automatski, a ne eksplicitno pomoću koda.) Slanje poruke od klijenta do servera pomuću UDP soketa je tako jednostavno! Posle slanja paketa, klijent čeka da primi podatke sa servera.


```
modifiedMessage,serverAddress=clientSocket.recvfrom(2048)
```

Kada paket stigne sa interneta u soket klijenta, pomoću gornjeg reda, podaci paketa se smeštaju u promenljivu modified Message, a izvorna adresa paketa u promenljivu server Address. Promenljiva server Address sadrži i IP adresu servera i broj porta servera. Programu UDPClient stvarno nije potrebna ova informacija o adresi servera, jer on već zna adresu servera od početka; međutim ovaj red u jeziku *Python* ipak pruža serveru adresu. Metoda recvfrom takođe uzima bafer veličine 2048 kao input. (Ova veličina bafera radi u većini slučajeva.)

```
printmodifiedMessage
```

Ovaj red štampa izmenjenu poruku na displeju korisnika. Trebalo bi da bude originalni red koji je korisnik ispisao, ali sada velikim slovima.

```
clientSocket.close()
```

Ovaj red zatvara soket. Proces se tada završava.

Program UDPServer.py

Pogledajmo sada serversku stranu aplikacije:

```
fromsocketimport*
serverPort=12000
serverSocket=socket(AF_INET,SOCK_DGRAM)
serverSocket.bind(('',serverPort))
print "The server is ready to receive"
while1:
      message,clientAddress=serverSocket.recvfrom(2048)
      modifiedMessage=message.upper()
      serverSocket.sendto(modifiedMessage,clientAddress)
```

Primetite da je početak programa UDPServer sličan programu UDPClient. On takođe uvozi modul soketa, postavlja celobrojnu promenljivu serverPort na 12000 i kreira soket tipa SOCK_DGRAM (UDP soket). Prvi red koda koji se značajno razlikuje od programa UDPClient je:

```
serverSocket.bind(('',serverPort))
```

Gornji red povezuje (tj. dodeljuje) broj porta 12000 soketu servera. Stoga, u kodu programa UDPServer (koji je napisao programer aplikacije) je jasno dodeljen broj porta soketu servera. Na ovaj način, kada neko pošalje paket na port 12000 na IP adresi servera, tada će paket biti usmeren ka ovom soketu. Program UDPServer tada

ulazi u petlju while; ova petlja dozvoljava programu UDPServer da neograničeno preuzima i obrađuje pakete od klijenata. U petlji while, program UDPServer čeka da paket stigne.

```
message,clientAddress=serverSocket.recvfrom(2048)
```

Ovaj red koda sličan je onom koji vidimo u programu UDPClient. Kada paket stigne u soket servera, podaci paketa se smeštaju u promenljivu message i izvorna adresa paketa se smešta u promenljivu client Address. Promenljiva client Address sadrži i IP adresu klijenta i broj porta klijenta. Ovde, *će* program UDPServer iskoristiti informacije o ovim adresama, jer to obezbeđuje povratnu adresu, slično povrtnoj adresi kod obične pošte. Pomoću ovih informacija o adresi izvora, server sada zna gde bi trebalo da usmeri svoj odgovor.

```
modifiedMessage=message.upper()
```

Ovaj red je srce naše jednostavne aplikacije. On preuzima red koji je klijent poslao i koristi metodu: „upper ()" da ga pretvori u velika slova.

```
serverSocket.sendto(modifiedMessage,clientAddress)
```

Ovaj poslednji red pripaja adresu klijenta (IP adresa i broj porta) poruci ispisanoj velikim slovima, i šalje, rezultujući paket u soket servera. (Kao što smo pomenuli ranije, adresa servera je takođe pripojena paketu, iako se ovo radi automatski, umesto da se eksplicitno radi pomoću koda.) Internet će tada isporučiti pakt na ovu adresu klijenta. Nakon što server pošalje paket, on ostaje u petlji while, čekajući da stigne još jedan UDP paket (od bilo kog klijenta koji se izvršava na bilo kom računaru).

Da bi testirali par programa instalirajte i kompajlirajte program UDPClient.py na jednom računaru, a program UDPServer.py na drugom računaru. Proverite da li ste uključili ispravan naziv računara ili IP adresu servera u programu UDPClient. py. Zatim, izvršićete program UDPServer.py, kompajlirani program servera, na računaru serveru. Ovim se kreira proces na serveru koji je u stanju mirovanja, sve dok ga neki klijent ne kontaktira. Tada na klijentu izvršavate program UDPClient.py, kompajlirani klijentski program. Ovim se kreira proces na klijentu. Konačno, da bi se aplikacija koristila kod klijenta, otkucajte rečenicu iza koje ide znak za početak novog reda.

Sopstvenu klijentsko-serverskuUDP aplikaciju možete razviti, tako što ćete za početak malo izmeniti klijentski ili serverski program. Na primer, umesto da pretvorite sva slova u velika, server može da izračuna koliko se puta slovo *s* pojavljuje i da vrati taj broj. A možete da izmenite klijentski program, tako da, nakon što preuzme rečenicu ispisanu velikim slovima, korisnik nastavi da šalje još rečenica serveru.

2.7.2 Programiranje soketa pomoću protokola TCP

Za razliku od protokola UDP, protokol TCP je protokol koji se zasniva na vezi. Ovo znači da, pre nego što klijent i server počnu da razmenjuju podatke, oni prvo moraju da se usaglase i uspostave TCP vezu. Jedan kraj TCP veze pripojen je soketu klijenta, a drugi soketu servera. Kada se kreira TCP veza , mi je povezujemo sa njenom adresom soketa klijenta (IP adresa i broj porta), i adresom soketa servera (IP adresa i broj porta). Kada se uspostavi TCP veza, ukoliko jedna strana želi da pošalje podatke drugoj strani, ona jednostavno spusti podatke kroz TCP vezu putem njenog soketa. Ovo je različito u odnosu na protokol UDP, za koji server mora da pripoji odredišnu adresu paketu, pre nego što ga spusti u soket.

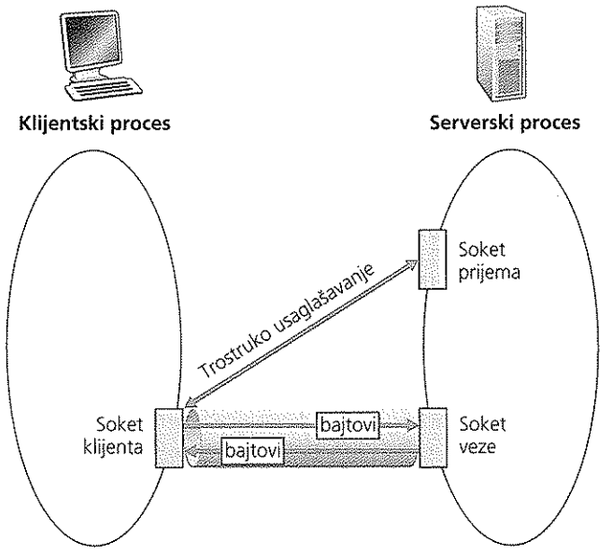
Pogledajmo sada izbliza kako izgleda interakcija između klijentskog i serverskog programa kod TCP protokola. Klijent ima zadatak da inicira kontakt sa serverom. Da bi mogao da odgovori na klijentov poziv, server mora da bude spreman, što podrazumeva dve stvari. Najpre, kao i u slučaju protokola UDP, TCP server mora da se izvršava kao proces, pre nego što klijent uopšte i pokuša da inicira kontakt. Drugo, serverski program mora imati neku vrstu vrata – tačnije specijalni soket – na kojem očekuje inicijalni kontakt od procesa klijenta koji se izvršava na bilo kom računaru. Vraćajući se za trenutak poređenju procesa i soketa sa kućom i vratima, ovaj klijentov inicijalni kontakt često se naziva „kucanje na vrata dobrodošlice".

Pošto se serverski proces izvršava, klijentski proces inicira TCP vezu do serverskog procesa. Ovo se postiže tako što se u klijentskom programu pravi TCP soket. Prilikom pravljenja TCP soketa klijent navodi adresu soketa dobrodošlice serverskog procesa, odnosno IP adresu servera i broj porta soketa. Nakon što se u klijentskom programu napravi soket, klijent inicira postupak trostrukog usaglašavanja i uspostavlja TCP vezu sa serverom. Trostruko usaglašavanje odigrava se na transportnom sloju i potpuno je nevidljivoi za klijentski i za serverski program.

Tokom trostrukog usaglašavanja, klijentski proces kuca na vrata dobrodošlice serverskog procesa. Kada „čuje" ovo kucanje, server pravi nova vrata – tačnije, *novi* soket – koji je namenjen samo tom klijentu. U primeru koji sledi, vrata dobrodošlice su objekat TCP soket koji zovemo `serverSocket;` novokreirani soket koji je dodeljen klijentu prilikom uspostvaljanja veze nazivamo `connectionSocket`. Studenti koji se po prvi put susreću sa TCP soketima ponekad mešaju soket dobrodošlice (koji je početna tačka kontakta za sve klijente koji čekaju na komunikaciju sa serverom) i svaki novokreirani soket veze serverske strane koji se redom kreira za komuniciranje sa svakim klijentom.

Iz perspektive aplikacije soket klijenta i soket veze servera su direktno povezani pomoću cevovoda. Kao što je prikazano na slici 2.29 klijentski proces može da pošalje bilo koji bajt u svoj soket, a TCP protokol garantuje da će serverski proces primiti (kroz soket veze) svaki bajt po redosledu slanja. Stoga, TCP protokol pruža pouzdanu uslugu između klijentskog i serverskog procesa. Pored toga, kao što ljudi mogu da ulaze i izlaze kroz ista vrata, klijentski proces, ne samo da šalje bajtove u soket, već ih i preuzima iz soketa; slično, serverski proces, ne samo da preuzima bajtove, već ih i šalje u soket veze.

Koristimo istu jednostavnu klijentsko-serversku aplikaciju kako bi prikazali porgramiranje soketa za protokol TCP: klijent šalje jedan red podataka serveru, server taj red pretvara u velika slova i šalje ga nazad klijentu. Slika 2.30 ističe glavne aktivnosti vezane za sokete klijenta i servera koji komuniciraju preko transportne usluge protokola TCP.

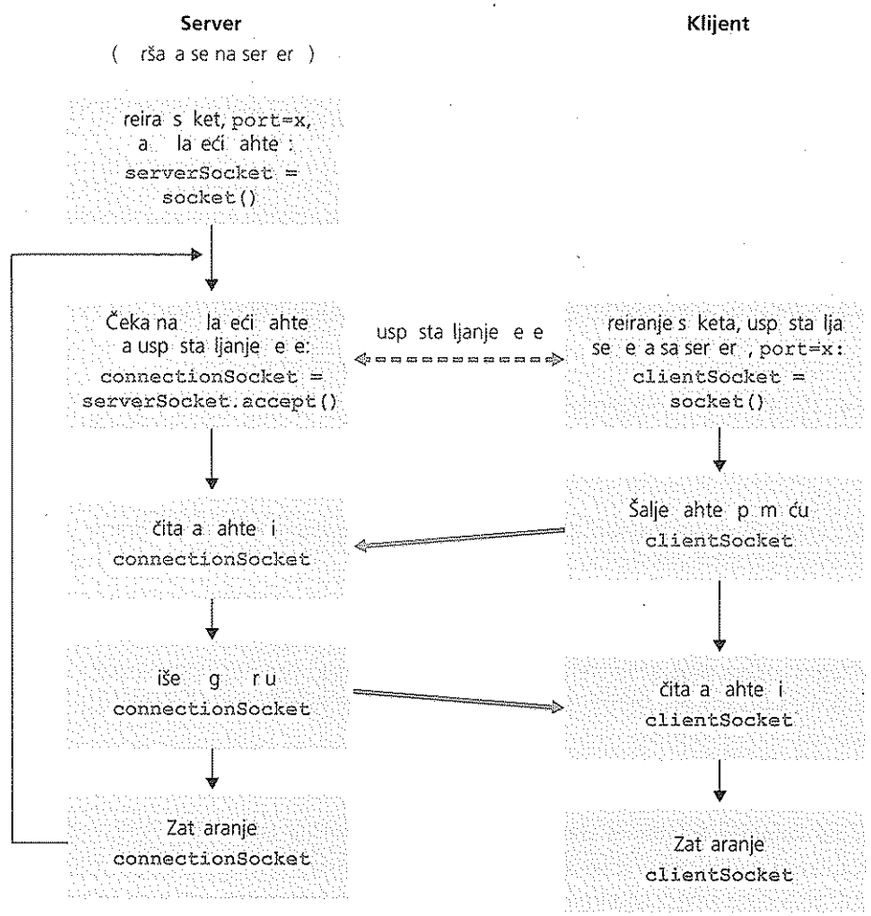


Slika 2.29 ◈ TCP Server proces ima dva soketa

Program TCPClient.py

Evo kako izgleda kôd klijentske strane aplikacije:

```
from socket import *
serverName='servername'
serverPort=12000
clientSocket=socket(AF_INET,SOCK_STREAM)
clientSocket.connect((serverName,serverPort))
sentence=raw_input('Input lower case sentence:')
clientSocket.send(sentence)
modifiedSentence=clientSocket.recv(1024)
print'FromServer:',modifiedSentence
clientSocket.close()
```

Slika 2.30 ◈ Klijentsko serverska aplikacija koja koristi protokol TCP

Pogledajmo sada različite redove koda koji se značajno razlikuju od UDP implementacije . Prvi takav red je kreiranje soketa klijenta.

```
clientSocket=socket(AF_INET,SOCK_STREAM)
```

Ovaj red kreira soket klijenta i zove se `clientSocket`. Prvi parametar ponovo ukazuje da korišćena mreža koristi IPv4. Drugi parametar ukazuje da je soket tipa `SOCK_STREAM`, što znači da je to soket protokola TCP (a ne soket protokola

UDP). Primetite da ponovo nismo naveli broj porta soketa klijenta kada ga kreiramo; umesto toga dozvolićemo operativnom sistemu da to uradi za nas. Sada je sledeći red koda veoma različit od onoga što smo videli u programu UDPClient:

```
clientSocket.connect((serverName,serverPort))
```

Setite se da, pre nego što klijent pošalje podatke serveru (ili obrnuto), koristeći soket protokola TCP, mora prvo da se uspostavi TCP veza između klijenta i servera. Red iznad inicira TCP vezu između klijenta i servera. Parametar metode: „`connect()`" je adresa serverske strane veze. Kada se ovaj red koda izvrši, izvršava se trostruko usaglašavanje, a između klijenta i servera se uspostavlja TCP veza.

```
sentence=raw_input('Inputlowercasesentence:')
```

Kao i kod programa UDPClient gornji red služi da se dobije rečenicu od korisnika. String sentence nastavlja da prikuplja karaktere, dok korsnik ne završi red kucanjem znaka za novi red. Sledeći red koda se takođe veoma razlikuje od programa UDPClient:

```
clientSocket.send(sentence)
```

Gornji red šalje string `sentence` kroz soket klijenta u TCP vezu. Primetite da program *ne* kreira eksplicitno paket i da ne pripaja odredišnu adresu paketu, kao u slučaju soketa protokola UDP. Umesto toga klijentski program jednostavno spusti bajtove stringa `sentence` u TCP vezu. Klijent zatim čeka da preuzme bajtove od servera.

```
modifiedSentence=clientSocket.recv(2048)
```

Kada karakteri stignu sa servera, smeštaju se u string `modified Sentence`. Karakteri nastavljaju da se akumuliraju u string-u `modified Sentence`, sve dok se red ne završi karakterom za novi red. Posle štampanja rečenice sa velikim slovima, zatvaramo soket klijenta:

```
clientSocket.close()
```

Ovaj poslednji red zatvara soket i, iz tog razloga, zatvara TCP vezu između klijenta i servera. To dovodi do toga da protokol TCP na klijentu, pošalje TCP poruku TCP protokolu na serveru (videti odeljak 3.5).

Program TCPServer.py

Pogledajmo sada kako izgleda serverski program.


```
fromsocketimport*
serverPort=12000
serverSocket=socket(AF_INET,SOCK_STREAM)
serverSocket.bind(('',serverPort))
serverSocket.listen(1)
print'The server is ready to receive'
while 1:
      connectionSocket,addr=serverSocket.accept()
      sentence=connectionSocket.recv(1024)
      capitalizedSentence=sentence.upper()
      connectionSocket.send(capitalizedSentence)
      connectionSocket.close()
```

Pogledajmo sada redove koji se značajno razlikuju od programa UDPServer i programa TCPClient. Kao i u programu TCPClient, server kreira soket protokola TCP:

```
serverSocket=socket(AF_INET,SOCK_STREAM)
```

Slično programu UDPServer, povezujemo broj porta servera `serverPort` sa ovim soketom:

```
serverSocket.bind(('',serverPort))
```

Međutim pomoću protokola TCP, `serverSocket` će biti naš soket dobrodošlice. Pošto uspostavimo vrata dobrodošlice, sačekaćemo i poslušati da li će neki klijent pokucati na vrata:

```
serverSocket.listen(1)
```

Ovaj red ima server koji osluškuje zahteve klijenta za uspostvaljanje TCP veze. Parametar navodi makismalan broj veza na čekanju (najmanje 1).

```
connectionSocket,addr=serverSocket.accept()
```

Kada klijent pokuca na vrata, program poziva metodu: „`accept ()`" za serverSocket, koja kreira novi soket na serveru pod nazivom `connectionSocket` (soket veze) koji se dodeljuje ovom određenom klijentu. Klijent i server tada završavaju usaglašavanje, kreirajući TCP vezu između klijentovog soketa – `clientSocket-a` i serverovog soketa – `connectionSocket-a`. Kada se uspostavi TCP veza, tada klijent i server putem veze mogu da šalju bajtove jedan drugom. Pomoću protokola TCP nije samo zagarantovano da će svi bajtovi, poslati s jedne strane, stići na drugu stranu, nego će garantovano stići i istim redosledom kojim su poslati.

```
connectionSocket.close()
```

U ovom programu posle slanja izmenjene rečenice klijentu, zatvaramo soket veze. Međutim, pošto soket servera – `serverSocket`, ostaje otvoren, još jedan klijent može sada da pokuca na vrata i pošalje serveru rečenicu da je izmeni.

Ovim se završava naša diskusija o programiranju soketa za protokol TCP. Ohrabreni ste da pokrenete ova dva programa na dva odvojena računara, a takođe i da ih izmenite, kako biste postigli ciljeve koji se malo razlikuju. Par UDP programa bi trebalo da poredite sa parom programa TCP i da vidite kako se razlikuju. Takođe, trebalo bi da uradite većinu zadataka vezanih za programiranje soketa, koji su opisani u poglavljima 2,4 i 7. Konačno, nadamo se da ćete jednog dana, kada ovladate ovim i mnogo naprednijim programima soketa, napisati svoju popularnu mrežnu aplikaciju, postati veoma bogati i slavni, i setiti se autora ovog udžbenika!

2.8 Rezime

U ovom poglavlju proučavali smo mrežne aplikacije – od zamisli do njihove realizacije. Saznali smo da je neizbežna klijentsko-serverska arhitektura prihvaćena u većini internet aplikacija i videli smo kako se ova arhitektura koristi kod protokola HTTP, FTP, SMTP, POP3 i DNS. Podrobno smo proučili ove, veoma važne protokole aplikativnog sloja i aplikacije koje ih koriste (veb, prenos datoteka, e-pošta i DNS). Takođe smo proučavali sve značajniju P2P arhitekturu i kako se ona koristi u mnogim aplikacijama. Istražili smo kako se API soket koristi za pravljenje mrežnih aplikacija. Prošli kroz sve faze korišćenja soketa za transportne usluge s kraja na kraj, sa uspostavljanjem veze (TCP) i bez uspostavljanja veze (UDP). Naš prvi korak u putovanju od vrha slojevite mrežne arhitekture ka njenom dnu ovim je završen!

Na samom početku knjige, u odeljku 1.1, ponudili smo pomalo nepreciznu, ogoljenu definiciju protokola: „format poruka i redosled po kome se te poruke razmenjuju između najmanje dva ili više eniteta koji međusobno komuniciraju, kao i postupke koji se preduzimaju posle slanja i/ili prijema određenih poruka ili nekog drugog događaja". Ono što smo u ovom poglavlju rekli o protokolima HTTP, FTP, SMTP, POP3 i DNS, dopunilo je smisao ove definicije. Protokoli predstavljaju ključni koncept umrežavanja; proučavanje protokola aplikativnog sloja omogućilo nam je da steknemo mnogo bolji osećaj o njihovom značaju.

U odeljku 2.1 opisali smo modele usluga koje protokoli TCP i UDP nude aplikacijama koje ih pozovu. Bliže smo upoznali te usluge, razvijajući jednostavne aplikacije koje koriste protokole TCP i UDP u odeljku 2.7. Međutim, nismo puno govorili o tome kako ovi protokoli obezbeđuju pomenute modele usluga. Na primer, znamo da TCP nudi uslugu pouzdane isporuke podataka, ali ništa nismo rekli o tome kako to radi. U sledećem poglavlju ćemo zato obratiti pažnju, ne samo na pitanje *šta* transportni protokoli rade, već *kako* i *zašto* to rade.

Naoružani znanjima o strukturi internet aplikacija i protokolima aplikativnog sloja, spremni smo za nastavak našeg puta naniže, kroz skup protokola i ispitivanje transportnog sloja u poglavlju 3.

Domaći zadaci: problemi i pitanja

Poglavlje 2: Kontrolna pitanja

ODELJAK 2.1

R1. Navedite pet javno dostupnih internet aplikacija i protokole aplikativnog sloja koje one koriste.

R2. U čemu je razlika između arhitekture mreže i arhitekture aplikacije?

R3. Kada par procesa međusobno komuniciraju, koji od ta dva procesa je klijent, a koji server?

R4. Da li se slažete sa sledećom izjavom u vezi sa P2P aplikacijom za deljenje datoteka ,: „Ne postoje pojmovi klijentska i serverska strana komunikacije". Obrazložite svoj odgovor.

R5. Pomoću kojih informacija proces koji se izvršava na jednom računaru prepoznaje proces koji se izvršava na drugom računaru?

R6. Pretpostavimo da želite da obavite prenos podatakasa udaljenog klijenta na server, što je brže moguće. Da li ćete koristiti protokol UDP ili protokol TCP? Zašto?

R7. Prema slici 2.4 vidimo da nijedna od navedenih aplikacija ne zahteva istovremeno i isporuku podataka bez gubitaka i ispunjenje vremenskih zahteva. Možete li da zamislite aplikaciju koja bi zahtevala isporuku podataka bez gubitaka i koja istovremeno zahteva pravovremenu isporuku?

R8. Navedite četiri opšte vrste usluga koje transportni protokoli nude. Za svaku od tih usluga navedite koju nudi protokol UDP, a koje protokol TCP (ili oba).

R9. Sećate se da se protokol TCP može poboljšati SSL uslugom kojom se obezbeđuje bezbednost između procesa, uključujući šifrovanje. Da li SSL radi na transportnom sloju ili na aplikativnom sloju? Ukoliko programer aplikacije želi da koristi protokol TCP poboljšan uslugom SSL, šta mora da uradi?

ODELJAK 2.2-2.5

R10. Šta znači protokol usaglašavanja (eng. handshaking)?

R11. Zbog čega se protokoli HTTP, FTP, SMTP i POP3 izvršavaju preko protokola TCP, a ne preko protokola UDP?

R12. Zamislite veb lokaciju za elektronsku trgovinu koja želi da čuva podatke o svakom svom korisniku. Opišite kako je to moguće učiniti pomoću kolačića.

R13. Opišite kako veb keširanje smanjuje kašnjenje pri dobijanju zahtevanog objekta. Da li veb keširanje smanjuje kašnjenje za sve zahtevane objekte od strane korisnika, ili samo za neke objekte? Zašto?

R14. Uspostavite Telnet sesiju sa nekim veb serverom i pošaljite poruku zahteva u više redova. U poruku zahteva ubacite red zaglavlja `If-modified-since:` kako biste u odgovoru dobili statusnu poruku: `304 Not Modified`.

R15. Zašto se kaže da protokol FTP šalje kontrolne informacije „izvan opsega"?

R16. Pretpostavimo da Alisa sa svog naloga za e-poštu, zasnovanom na vebu (recimo Hotmail ili gmail), želi da pošalje poruku Bobu koji svom serveru za e-poštu pristupa preko protokola POP3. Opišite put ove poruke od Alisinog do Bobovog računara. Obavezno navedite sve protokole aplikativnog sloja koji se koriste za prenos poruke sa jednog na drugi računar.

R17. Odštampajte zaglavlje neke e-poruke koju ste nedavno primili. Koliko puta se ponavlja red zaglavlja `Received:` u njoj? Analizirajte sve te redove zaglavlja u poruci.

R18. U čemu je, sa stanovišta korisnika, razlika između režima rada preuzimanja i brisanja i režima rada preuzimanja i zadržavanja protokola POP3?

R19. Da li je moguće da veb server i server za e-poštu neke organizacije imaju istovetan pseudonim za imena svojih računara (recimo `foo.com`)? Kog tipa je RR zapis u kome se nalazi ime računara servera za e-poštu?

R20. Pregledajte primljene poruke e-pošte i ispitajte zaglavlje poruke poslate od korisnika čija je adresa e-pošte sadrži .edu. Da li je moguće na osnovu zaglavlja odrediti IP adresu računara sa kog je poruka poslata? Uradite isto i za poruku poslatu sa gmail naloga.

ODELJAK 2.6

R21. Kod protokola BitTorrent, pretpostavimo da Alisa svakih 30 sekundi Bobu prosleđuje odsečke neke datoteke. Da li Bob obavezno mora da uzvrati istom merom i da Alisi vraća odsečke neke datoteke istom brzinom? Obrazložite.

R22. Uzmimo kao primer novi ravnopravni računar, Alisu, koja se pridružila grupi BitTorrent bez poseda odsečaka datoteka. Pošto nema odsečaka, ne može da postane jedna od četiri najbolja pošiljaoca za bilo koji drugi ravnopravni računar, jer nema šta da prenese na mrežu. Kako će onda Alisa da dobije svoj prvi odsečak?

R23. Šta je preklopljena mreža? Da li su ruteri njen sastavni deo? Šta su grane u preklopljenoj mreži?

R24. Razmotrite distribuiranu heš tabelu sa mrežastom topologijom (tj. svaki ravnopravni računar prati sve ravnopravne računare u sistemu). Koje su prednosti, a šta su nedostaci ovakvog dizajna? Navedite prednosti i nedostatke kružne distribuirane heš tabele (bez prečica)?

R25. Navedite barem četiri aplikacije kojima prirodno odgovara P2P arhitektura. (Pomoć: Distribucija datoteka i instant razmena poruka su dve).

ODELJAK 2.7

R26. UDP serveru koji je opisan u odeljku 2.7 bio je potreban samo jedan soket, dok su TCP serveru bila potrebna dva. Zašto? Kada bi TCP server morao da podrži *n* istovremenih veza, od kojih je svaka sa različitog računara klijenta, koliko bi mu soketa za to bilo potrebno?

R27. Zbog čega u klijentsko-serverskoj aplikaciji, koja koristi protokol TCP iz odeljka 2.7 serverski, program mora da se pokrene pre klijentskog? Zbog čega u klijentsko-serverskoj aplikaciji za protokol UDP klijentski program može da se pokrene pre serverskog?

Problemi

P1. Tačno ili netačno?

a. Korisnik zahteva veb stranu koja se sastoji od nekog teksta i tri slike. Za ovu stranu klijent šalje jednu poruku zahteva, a prima četiri poruke odgovora.

b. Dve različite veb strane (na primer, `www.mit.edu/research.html` i `www.mit.edu/students.html`) mogu da se pošalju istom postojanom vezom.

c. Sa nepostojanom vezom između pretraživača i izvornog servera moguće je da jedan TCP segment prenosi dve različite HTTP poruke zahteva.

d. Zaglavlje `Date:` u HTTP poruci odgovora pokazuje kada je objekat koji se nalazi u tom odgovoru poslednji put izmenjen.

e. HTTP poruka odgovora nikad nema prazno telo poruke.

P2. Pročitajte RFC dokument 959 koji se odnosi na protokol FTP. Navedite sve klijentske komande koje su podržane ovim dokumentom.

P3. Zamislite HTTP klijenta koji želi da preuzme veb dokument sa određene URL adrese. IP adresa odgovarajućeg HTTP servera je u početku nepoznata. Koji su protokoli transportnog i aplikativnog sloja, pored protokola HTTP, potrebni u ovom slučaju?

P4. Posmatramo sledeći niz ASCII znakova koje je pokupio program *Wireshark* pošto je pretraživač posalo HTTP GET poruku (u stvari, ovo je stvarni sadržaj HTTP GET poruke). Karakteri *<cr><lf>* su znakovi za novi red i povratak na početak reda (odnosno kurzivom ispisani znak *<cr>* u tekstu koji sledi predstavlja znak za novi red koji se nalazi na tom mestu u zaglavlju HTTP poruke). Odgovorite ne sledeća pitanja, navodeći gde ste u HTTP GET poruci pronašli odgovor.

```
GET/cs453/index.htmlHTTP/1.1<cr><lf>Host:gai
a.cs.umass.edu<cr><lf>User-Agent:Mozilla/5.0(
Windows;U;WindowsNT5.1;en-US;rv:1.7.2)
Gecko/20040804Netscape/7.2(ax)<cr><lf>Accept:ext/
xml,application/xml,application/xhtml+xml,text
/html;q=0.9,text/plain;q=0.8,image/png,*/*;q=0.5
<cr><lf>Accept-Language:en-us,en;q=0.5<cr><lf>Accept-
Encoding:zip,deflate<cr><lf>Accept-Charset:ISO
-8859-1,utf-8;q=0.7,*;q=0.7<cr><lf>Keep-Alive:300<cr>
<lf>Connection:keep-alive<cr><lf><cr><lf>
```

a. Koja je URL adresa dokumenta koju zahteva pretraživač?

b. Koju verziju protokola HTTP koristi pretraživač?

c. Da li pretraživač traži nepostojanu ili postojanu vezu?

d. Koja je IP adresa računara na kome se izvršava pretraživač?

e. Koji tip pretraživača započinje ovu poruku? Zašto je potreban tip pretraživača u HTTP poruci zahteva?

P5. Tekst u nastavku pokazuje odgovor koji je poslao server na HTTP GET poruku iz prethodnog pitanja. Odgovorite ne sledeća pitanja, navodeći gde ste u poruci ispod, pronašli odgovor.

```
HTTP/1.1200OK<cr><lf>Date:Tue,07Mar2008
12:39:45GMT<cr><lf>Server:Apache/2.0.52(Fedora)
<cr><lf>Last-Modified:Sat,10Dec200518:27:46
GMT<cr><lf>ETag:"526c3-f22-a88a4c80"<cr><lf>Accept-
Ranges:bytes<cr><lf>Content-Length:3874<cr><lf>
Keep-Alive:timeout=max=100<cr><lf>Connection:
Keep-Alive<cr><lf>Content-Type:text/html;charset=
ISO-8859-1<cr><lf><cr><lf><!doctypehtmlpublic"-
//w3c//dtd html 4.0 transitional//en"><lf><html><lf>
<head><lf><metahttp-equiv="Content-Type"
content="text/html;charset=iso-8859-1"><lf>
<metaname="GENERATOR"content="Mozilla/4.79[en]
(WindowsNT5.0;U)Netscape]"><lf><title>CMPSCI453/591/
NTU-ST550ASpring2005homepage</title><lf></head><lf>
<muchmoredocumenttextfollowinghere(notshown)>
```

a. Da li je server uspešno pronašao odgovarajući dokument ili nije? Kada je odgovor sa dokumentom obezbeđen?

b. Kada je dokument poslednji put menjan?

c. Koliko bajtova ima dokument koji se vraća?

d. Koji su prvih 5 bajtova dokumenta koji se vraća u odgovoru? Da li se server složio da uspostavi postojanu vezu?

P6. Pronađite specifikacije za protokol HTTP/1.1 (RFC 2616). Odgovorite na sledeća pitanja:

a. Objasnite mehanizme koji se koriste za signalizaciju između klijenta i servera u obaveštavanju da je postojana veza prekinuta. Da li vezu može da prekine klijent, server, ili bilo ko od njih?

b. Koje usluge za šifrovanje obezbeđuje protokol HTTP?

c. Da li klijent može da uspostavi tri ili više veza istovremeno sa određenim serverom?

d. Server ili klijent mogu da zatvore transportnu vezu između njih, ukoliko jedan od njih utvrdi da je veza u stanju mirovanja neko vreme. Da li je moguće da jedna strana počinje da prekida vezu dok druga strana preuzima podatke putem te veze? Objasnite.

P7. Pretpostavimo da ste u svom veb pretraživaču mišem pritisnuli link, da biste dobili neku veb stranu. IP adresa odgovarajuće URL adrese nije keširana na vašem lokalnom računaru, tako da je za njeno pribavljanje neophodno pokretanje DNS upita. Pretpostavimo da je posećeno n DNS servera pre nego što je vaš računar dobio IP adresu od DNS-a; za sve ove posete potrebno je RTT vreme koje se sastoji od , ..., vremena. Dalje, pretpostavimo da ova veb strana na koju upućuje ovaj link sadrži samo jedan objekat koji sačinjava malo HTML teksta. Neka označava RTT vreme između lokalnog računara i servera na kome se ovaj objekat nalazi. Ukoliko pretpostavimo da je trajanje prenosa objekta nula, koliko vremena protekne od trenutka kada korisnik izabere link do trenutka kada klijent primi traženi objekat?

P8. Pozivajući se na problem 7, pretpostavimo da su u HTML datoteci referencirana tri veoma mala objekta na istom serveru. Ako zanemarimo trajanje prenosa, koliko vremena protekne u slučaju:

a. Nepostojanih HTTP veza bez paralelnih TCP veza?

b. Nepostojanih HTTP veza sa pretraživačem konfigurisanim za 5 paralelnih TCP veza?

c. Postojanih HTTP veza?

P9. Vratimo se na sliku 2.12 na kojoj je prikazana mreža neke institucije povezana na internet. Pretpostavimo da je prosečna veličina objekta 85 0000 bitova i da prosečna učestalost zahteva koje pretraživači unutar ove institucije šalju izvornim serverima iznosi 16 zahteva u sekundi. Takođe, pretpostavimo da prosečno proteknu tri sekunde, od trenutka kada ruter na internet strani pristupnog linka prosledi HTTP zahtev do trenutka kada primi odgovor (odeljak 2.2.5). Ukupno prosečno trajanje odziva predstavite kao sumu prosečnog kašnjenja pristupa (odnosno kašnjenje od rutera na internetu do rutera institucije) i prosečnog kašnjenja na internetu. Za prosečno kašnjenje pristupa koristite $\Delta/(1 - \Delta\beta)$, gde Δ predstavlja prosečno trajanje slanja objekta pristupnim linkom, a β brzinu pristizanja objekata na pristupni link.

a. Odredite ukupno prosečno trajanje odziva.

b. Sada pretpostavimo da je u lokalnoj mreži institucije instalirano keširanje i da je udeo uspešno pronađenih objekata u privremenoj memoriji 0,4. Odredite ukupno trajanje odziva.

P10. Uzmimo za primer kraći link dužine 10 m, preko koga pošiljalac može da prenosi brzinom od 150 b/s u oba smera. Pretpostavimo da su paketi koji sadrže podatke dužine 100 000 bitova i da su paketi koji sadrže samo kontrolne podatke (npr. ACK ili usaglašavanje) dužine 200 bitova. Pretpostavimo da postoji *N* paralelnih veza i da svaka dobija *1/N* deo propusnog opsega linka. Razmatramo korišćenje protokola HTTP i pretpostavimo da su svi preuzeti objekti dužine 100 kb, a da početni preuzeti objekat sadrži 10 referenci objekata na istom pošiljaocu. Da li bi paralelno preuzimanje preko paralelno uspostavljenih nepostojanih HTTP veza imalo smisla u ovom slučaju? Sada razmotrite korišćenje postojane HTTP veze. Da li očekujete značajnije poboljšanje u odnosu na nepostojanu vezu? Obrazložite i objasnite svoj odgovor.

P11. Obratite pažnju na prethodni problem. Pretpostavimo sada da link dele Bob i još četiri korisnika. Bob koristi paralelne primerke nepostojane HTTP veze, a ostala četiri korisnika koriste nepostojanu HTTP vezu bez paralelnih preuzimanja.

a. Da li paralelne veze pomažu Bobu da veb strane dobije brže? Zašto da, ili zašto ne?

b. Ako svih pet korisnika otvori pet paralelnih primeraka nepostojane HTTP veze, da li će i tada Bobove paralelne veze biti od koristi? Zašto da, ili zašto ne?

P12. Napišite jednostavan TCP program za server koji prihvata nekoliko redova teksta od klijenta i prikazuje ih na standardnom izlazu servera. (Ovo možete da uradite i menjanjem programa *TCPServer*.py koji smo naveli u tekstu.) Kompajlirajte i pokrenite svoj program. Zatim na bilo kom drugom računaru sa veb pretraživačem, podesite da proksi server pretraživača bude server na kome se izvršava vaš program; takođe, konfigurišite odgovarajući broj porta. Vaš pretraživač bi nakon toga trebalo da šalje svoje GET poruke zahteva vašem serveru, na čijem standardnom izlazu bi trebalo da se prikazuju te poruke. Koristeći ovako uspostavljenu platformu, ustanovite da li vaš pretraživač generiše uslovne GET poruke za objekte koji se nalaze u lokalnoj keš memoriji.

P13. U čemu je razlika između zaglavlja `MAIL FROM:` u protokolu SMTP i zaglavlja `From:` u samoj poruci?

P14. Kako protokol SMTP obeležava kraj tela poruke? A kako protokol HTTP? Da li za obeležavanje kraja tela poruke protokol HTTP može da koristi istu metodu kao i protokol SMTP? Objasnite.

P15. Pročitajte dokument RFC 5321 za protokol SMTP. Šta predstavlja MTA? Obratite pažnju na sledeću primljenu, nepoželjnu (to jest spam) poštu (koja je izmenjena, prava spam pošta). Pretpostavljajući da je pošiljalac ove nepoželjne poruke neprijateljski, a da su svi ostali računari iskreni, identifikujte neprijateljski računar host, koji je kreirao ovu nepoželjnu poruku.

```
From-FriNov0713:41:302008
Return-Path:<tennis5@pp33head.com>
Received:frombarmail.cs.umass.edu
(barmail.cs.umass.edu[128.119.240.3])by cs.umass.edu
(8.13.1/8.12.6)for<hg@cs.umass.edu>;Fri,7Nov2008
13:27:10-0500
```



```
Received:fromasusus-4b96 (localhost[127.0.0.1]) by
barmail.cs.umass.edu(SpamFirewall)for
<hg@cs.umass.edu>;Fri, 7Nov200813:27:07-0500
(EST)
Received:fromasusus-4b96([58.88.21.177])by
barmail.cs.umass.edufor<hg@cs.umass.edu>;Fri,
07 Nov 2008 13:27:07 -0500 (EST)
Received:from[58.88.21.177] by
inbnd55.exchangeddd.com;Sat, 8 Nov 2008 01:27:07+0700
From:"Jonny"<tennis5@pp33head.com>
To:<hg@cs.umass.edu>
Subject:Howtosecureyoursavings
```

P16. Pročitajte RFC dokument 1939 koji se odnosi na protokol POP3. Čemu služi komanda UIDL protokola POP3?

P17. Zamislite da bi trebalo da proverite svoju e-poštu putem protokola POP3.

a. Pretpostavimo da ste POP klijent za e-poštu podesili tako da radi u režimu preuzimanja i brisanja poruka. Dopunite sledeću transakciju:

C: `list`
S: `1498`
S: `2912`
S: `.`
C: `retr1`
S: `blahblah...`
S: `.........blah`
S: `.`
?
?

b. Pretpostavimo da ste POP klijent za e-poštu podesili tako da radi u režimu preuzimanja i zadržavanja poruka. Dopunite sledeću transakciju:

C: `list`
S: `1498`
S: `2912`
S: `.`
C: `retr1`
S: `blahblah...`
S: `.........blah`
S: `.`
?
?

c. Pretpostavimo da ste POP klijent za e-poštu podesili tako da radi u režimu preuzimanja i zadržavanja poruka. Koristeći ono što ste dopisali u delu (b), pretpostavimo da ste primili poruke 1 i 2, izašli iz POP klijenta i da ste pet minuta kasnije ponovo pokrenuli POP klijenta, kako biste preuzeli nove e-poruke. Pretpostavimo da u tih pet minuta niste dobili nijednu novu poruku. Ispišite transcript nove POP3 sesije.

P18. a.Šta je *whois* baza podataka?

b. Pomoću raznih whois baza podataka na internetu pronađite imena dva DNS servera. Navedite koje ste whois baze podataka koristili.

c. Pomoću programa *nslookup* sa svog lokalnog računara pošaljite DNS upite ka tri DNS servera – svom lokalnom DNS serveru i DNS serverima koje ste pronašli u delu (b). Pokušajte da ispitate izveštaje za upite tipa A, NS i MX. Ukratko opišite ono što ste dobili.

d. Programom *nslookup* pronađite veb server koji ima više IP adresa. Da li veb server vaše institucije (škole ili kompanije) ima više IP adresa?

e. Pomoću whois baze podataka ARIN odredite opseg IP adresa koji koristi vaša institucija.

f. Opišite način na koji napadač pomoću whois baza podataka i alatke *nslookup* može da prikupi obaveštenja o instituciji pre samog napada.

g. Obrazložite opravdanost javne dostupnosti whois baza podataka.

P19. U ovom probelmu, upotrebljavamo korisnu alatku *dig*, raspoloživu na *Unix* i *Linux* računarima za ispitivanje hijerarhije DNS servera. Setite se slike 2.21 na kojoj viši DNS server u DNS hijerarhiji delegira DNS upit na DNS server koji je niži hijerarhijski, vraćajući DNS klijentu naziv tog DNS servera nižeg reda. Prvo pročitajte glavnu stranu za alatku *dig*, a zatim odgovorite na sledeća pitanja.

a. Počevši od korenskog DNS servera (od jednog od korenskih servera [a-m].root.servers.net), započnite niz upita za IP adrese za veb server vašeg odseka, koristeći alatku *dig*. Prikažite listu imena DNS servera u lancu delegiranja prilikom odgovora na vaš upit.

b. Ponovite deo a) za nekoliko popularnih veb lokacija, kao što su google.com, yahoo.com ili amazon.com.

P20. Pretpostavimo da želite da pristupite keš memoriji lokalnog DNS servera vašeg odseka. Možete li grubo da odredite veb servere (izvan vašeg odseka) koji su najpopularniji među korisnicima vašeg odseka? Objasnite.

P21. Pretpostavimo da vaš odsek ima lokalni DNS server za sve računare u odseku. Vi ste običan korisnik (odnosno niste administrator mreže/sistema). Možete li da utvrdite da li se pre nekoliko sekundi pristupilo spoljnoj veb lokaciji sa računara iz vašeg odseka? Objasnite.

P22. Razmotrimo distribuiranje fajla $F = 15$ Gb na N ravnopravnihračunara. Uzvodna brzina servera je $u_s = 20$ Mb/s, a svi ravnopravni računari imaju nizvodnu brzinu $d_i = 2$ Mb/s i uzvodnu brzinu u. Za $N = 10$, 100 i 100 i u = 300 kb/s, 700 kb/s i 2 Mb/s napravite dijagram na kome su data najmanje vreme distribucije za sve kombinacije N i u za distribuciju, korišćenjem klijentsko-serverske arhitekture i P2P arhitekture.

P23. Posmatramo distribuiranje fajla od F bitova na$N$ računara, korišćenjem klijentsko-serverske arhitekture. Pretpostavimo da imamo prilagodljiv server koji može da istovremeno prenosi različitim brzinama do više ravnopravnih računara, pod uslovom da zbirna brzina nije veća od u_s.

a. Pretpostavimo da je $u_s/N \le d_{\min}$. Odredite način distribucije za koji je vreme distribucije NF/u_s.

b. Pretpostavimo da je $u_s/N \ge d_{\min}$. Odredite način distribucije za koji je vreme distribucije $F/d_{\min}$.

c. Zaključak je da je najkraće vreme distribucije u opštem slučaju $\max\{NF/u_s, F/d_{\min}\}$.

P24. Posmatramo distribuiranje fajla od F bitova na N računara, korišćenjem P2P arhitekture. Pretpostavimo da imamo prilagodljiv model. Zbog jednostavnosti, pretpostavimo da je $d$min veoma veliko, tako da propusna moć ravnopravnog računara za preuzimanje nikad nije usko grlo.

a. Pretpostavimo da je $u_s \le (u_s + u_1 + \ldots + u_N)/N$. Odredite način distribucije za koji je vreme distribucije F/u_s.

b. Pretpostavimo da je $u_s \ge (u_s + u_1 + \ldots + u_N)/N$. Odredite način distribucije za koji je vreme distribucije $NF/(u_s + u_1 + \ldots + u_N)$.

c. Zaključak je da je najkraće vreme distribucije u opštem slučaju $\max\{F/u_s, NF/(u_s + u_1 + \ldots + u_N)\}$.

P25. Posmatramo prekloplјenu mrežu koju sačinjava N računara, pri čemu je između svih parova računara uspostavljena TCP veza. Pored toga, pretpostavimo da ove TCP veze prolaze kroz ukupno M rutera. Koliko čvorova i grana postoji u ovakvoj prekloplјenoj mreži?

P26. Pretpostavimo da se Bob priključio BitTorrent torentu, ali ne želi da predaje podatke ostalim ravnopravnim računarima (tzv. slobodni jahač).

a. Bob tvrdi da može da preuzme kompletnu kopiju datoteke koja se deli u tom krugu. Da li je Bobova tvrdnja moguća? Zašto jeste, a zašto nije?

b. Bob dalje tvrdi da ovo „slobodno jahanje" može biti efikasnije, ako koristi kolekciju više računara (sa različitim IP adresama) u računarskoj laboratoriji njegovog odseka. Kako on to može da uradi?

P27. U kružnoj DHT u odeljku 2.6.2 pretpostavimo da je ravnopravni računar 3 shvatio da je upareni računar 5 otišao. Na koji način računar 3 može da ažurira informacije o stanju svog sledbenika? Koji ravnopravni računar je sada njegov prvi sledbenik? Koji je njegov drugi sledbenik?

Video napomena: Šetnja kroz distribuiranu heš tabelu

P28. U primeru o kružnom DHT u odeljku 2.6.2 pretpostavimo da novi ravnopravni računar 6 želi da se priključi distribuiranoj heš tabeli, a da i računar 6 samo zna samo IP adresu računara 15. Koji su koraci preduzeti?

P29. S obzirom da ceo broj $[0, 2^n - 1]$ može da se izrazi kao n-bitni binarni broj u DHT, svaki ključ može da se prestavi kao $k = (k_0, k_1, \ldots, k_{n-1})$, a svaki identifikator ravnopravnog računara prikazan je kao $p = (p_0, p_1, \ldots, p_{n-1})$. Definišimo sada XOR udaljenost između ključa k i uparenog računara p kao

$$d(k, p) = \sum_{j=0}^{n-1} |k_j - p_j| 2^j$$

Objasnite kako ova metrika može da se upotrebi za dodeljivanje (ključ, vrednost) parova ravnopravnim računarima. (Ako želite da naučite kako da gradite efikasne DHT pomoću prirodne metrike, pogledajte rad [Maymounkov 2002] u kome je opisana *Kademlia* distribuirana heš tabela.)

P30. S obzirom da su distribuirane heš tabele preklopljene mreže, one se neće nužno podudarati sa upotrebljenom fizičkom mrežom, jer dva susedna ravnopravna računara mogu fizički da budu jako udaljeni; na primer, jedan računar može da bude u Aziji, a njegov sused može da se nalazi u Severnoj Americi. Ako nasumično i na isti način dodelimo identifikatore novopridošlim, ravnopravnim računarima, da li će ova šema dodeljivanja prouzorkovati ovakvo nepoklapanje? Objasnite. I kako će ovakvo nepoklapanje uticati na učinak DHT-a?

P31. Instalirajte i kompajlirajte *Python* programe TCPClient i UDPClient na jednom računaru, a TCPServer i UDPServer na drugom.

a. Pretpostavimo da program TCPClient izvršavate pre izvršavanja programa TCPServer. Šta se dešava? Zašto?

b. Pretpostavimo da program UDPClient izvršavate pre izvršavanja programa UDPServer. Šta se dešava? Zašto?

c. Šta se dešava, ako koristite različite brojeve portova za klijentsku i serversku stranu?

P32. Pretpostavimo da u programu UDPClient.py, nakon što kreiramo soket, dodamo još jedan red:

```
clientSocket.bind(('',5432))
```

Da li će biti neophodno da promenite program UDPServer.py? Koji su brojevi portova za sokete u programu UDPClient i UDPServer? Koji su to brojevi bili pre ove izmene?

P33. Da li možete da konfigurišete vaš veb pretraživač, tako da otvori više istovremenih veza ka veb lokaciji? Koje su prednosti i nedostaci većeg broja istovremenih TCP veza?

P34. Videli smo da se internet TCP soketi bave podacima koji se šalju kao tok bajtova, ali UDP soketi prepoznaju granice poruke. Navedite jednu prednost i jedan nedostatak API-a koji se zasniva na bajtovima, u odnosu na API koji jasno prepoznaje i čuva granice poruke koju šalju aplikacije?

P35. Šta je Apache veb server? Koliko košta? Koje osobine trenutno poseduje? Možete potražiti na „Vikipediji" odgovor na ovo pitanje.

P36. Mnogi BitTorrent klijenti koirste distribuirane heš tabele za kreiranje distribuiranog tragača. Šta je „ključ", a šta „vrednost" za ove distribuirane tabele?

Zadaci za programiranje soketa

Prateća veb lokacija uključuje šest zadataka za programiranje soketa. Prva četiri zadatka prikazana su sažeto u nastavku teksta. Peti zadatak se tiče upotrebe protokola ICPM i sažeto je dat na kraju poglavlja 4. Šesti zadatak se bavi multimedijalnim protokolima i ukratko je prikazan na kraju poglavlja 7. Studentima toplo preporučujemo da urade nekoliko, ako ne i sve ove zadatke. Na veb lokaciji http://ww.awl.com/kurose-rose studenti mogu naći ove zadatke detaljno prikazane, kao i važne odlomke koda u programskom jeziku *Python*.

Zadatak 1: Veb server

U ovoj vežbi ćete razviti jednostavan veb server u programskom jeziku *Python* koji može da obradi samo jedan zahtev. Vaš veb server će (I) kreirati soket veze, kada ga klijent kontaktira (pretraživač); (II) preuzeće HTTP zahtev iz ove veze; (III) raščlaniti zahtev, kako bi se utvrdila zahtevana datoteka; (IV) dobiti zahtevanu datoteku iz sistema datoteka servera; (V) kreirati HTTP poruku odgovora, koja sadrži zahtevanu datoteku kojoj prethode redovi zaglavlja; i (VI) poslaće odgovor putem TCP veze pretraživaču koji šalje upit. Ukoliko pretraživač zahteva datoteku koja se ne nalazi na vašem serveru, server bi trebalo da vrati poruku o greški „404 Not Found (404 nije pronađen)".

Na pratećoj veb lokaciji obezbeđujemo osnovnu strukturu koda za vaš server. Vi bi trebalo da kompletirate kôd, pokrenete server i potom testirate server, slanjem zahteva pomoću pretražavača koji se izvršavaju na različitim računarima. Ako server pokrećete na računaru na kome se već izvršava neki veb server, tada bi na vašem veb serveru trebalo da upotrebite port koji je različit od porta 80.

Zadatak 2: UDP Pinger

U ovom zadatku iz programiranja pisaćete program ping klijenta u programskom jeziku *Python*. Vaš klijent će poslati jednostavnu ping poruku serveru, preuzeće odgovarajuću pong poruku od servera, i utvrditi kašnjenje između vremena kada je poslata ping poruka i vremena kada je preuzeta pong poruka. Ovo kašnjenje naziva se vreme povratnog puta (RTT – Round Trip Time). Funkcionalnost koja je obez-

bađena od strane klijenta i servera slična je funkcionalnosti koju pruža standardni ping program koji se nalazi u modernim operativnim sistemima. Međutim, standardni ping programi koriste protokol ICMP, protokol za slanje kontrolnih poruka na Internetu (Internt Control Message Protocol), koji ćemo raditi u poglavlju 4. Ovde ćemo kreirati nestandardni (ali jednostavan!) ping program koji se zasniva na protokolu UDP.

Vaš ping program bi putem protokola UDP trebalo da pošalje 10 ping poruka ciljanom serveru., Kada se vrati odgovarajuća pong poruka, vaš klijent bi za svaku poruku trebalo da odredi i odštampa RTT. Pošto UDP nije pouzdan protokol, paket koji je poslao klijent ili server može da se izgubi. Iz tog razloga, klijent ne može beskonačno da čeka na odgovor ping poruci. Klijent bi trebalo da sačeka sekundu na odgovor od servera; ako se odgovor ne dobije, klijent bi trebalo da pretpostavi da je paket izgubljen i stoga da odštampa odgovarajuću poruku.

U ovom zadatku ćete dobiti ceo kôd za server (raspoloživ na pratećoj veb lokaciji). Vaš zadatak je da napišete klijentski kôd, koji će biti veoma sličan serverskom kodu. Preporučujemo da prvo pažljivo proučite serverski kôd. Tada možete da napišete klijentski kôd, i slobodno isečete i umetnete redove iz serverskog koda.

Zadatak 3: Klijent za e-poštu

Cilj ovog programerskog zadatka je da kreirate jednostavan klijent za e-poštu koji šalje e-poštu svakom primaocu. Vaš klijent bi trebalo da uspostavi TCP vezu sa serverom e-pošte (npr. *Google* server e-pošte), komunicira sa serverom e-pošte pomoću protokola SMTP, šalje poruku e-pošte primaocu (npr. vaš prijatelj) putem servera e-pošte i konačno prekida TCP vezu sa serverom e-pošte.

Za ovaj zadatak, prateća veb lokacija pruža osnovnu strukturu koda vašeg klijenta. Vaš zadatak je da završite kôd i slanjem e-pošte na različite korisničke naloge testirate klijenta. Takođe, možete da pokušate da pošaljete preko različitih servera (npr, preko *Google* servera e-pošte i preko servera e-pošte vašeg univerziteta).

Zadatak 4: Višenitni veb proksi server

U ovom zadatku razvićete veb proksi server. Kada vaš proksi server primi HTTP zahtev za neki objekat od pretraživača, on stvara novi HTTP zahtev za isti objekat i šalje ga izvornom serveru. Kada proksi server preuzme odgovarajući HTTP odgovor sa objektom sa izvornog servera, on, uključujući objekat, kreira novi HTTP odgovor i šalje ga klijentu. Ovaj proksi server biće višenitni, tako da će moći da upravlja sa više zahteva u isto vreme.

Za ovaj zadatak, prateća veb lokacija pruža osnovnu strukturu koda za proksi server. Vaš zadatak je da završite kôd i da ga onda testirate, tako što ćete pomoću različitih pretraživača zahtevati objekte na vebu preko vašeg proksi servera.

Wireshark laboratorijska vežba: protokol HTTP

a i k s ih *Wireshark* la rat rijskih ež i, etaljan prika ih ež i nala i se na e l ka iji knjige http: www.awl. m kur se-r ss.

Video napomena

Pošto smo se u laboratorijskoj vežbi 1 upoznali sa alatom za nadgledanje protoka paketa, programom *Wireshark*, sada smo spremni da upotrebimo program *Wireshark* za ispitivanje protokola na delu. U ovoj laboratorijskoj vežbi, istražićemo nekoliko aspekata protokola HTTP: osnovnu GET/reply interakciju, formate HTTP poruke, preuzimanje velikih HTTP datoteka, preuzimanje HTML datoteka sa ugrađenim URL adresama, postojane i nepostojane veze i HTTP proveru identiteta i bezbednost.

Wireshark laboratorijska vežba: protokol DNS

U ovoj vežbi izbliza ćemo analizirati klijentsku stranu protokola DNS koji nazive računara na internetu prevodi u IP adrese. Sećate se iz odeljka 2.5 da je uloga klijenta u sistemu DNS relativno jednostavna – on šalje upit svom lokalnom DNS serveru i od njega prima odgovor. Većina onoga što se događa u sistemu DNS, nije vidljivo je za DNS klijenta, zato što hijerarhijski DNS serveri međusobno komuniciraju i rekurzivno ili iterativno rešavaju dobijeni DNS upit. Iz perspektive DNS klijenta ovaj protokol je sasvim jednostavan – lokalnom DNS serveru se šalje upit i zatim se od njega očekuje odgovor. Dakle, u ovoj laboratorijskoj vežbi pratićemo rad protokola DNS.

Podroban opis svih *Wireshark* laboratorijskih vežbi pronaći ćete na pratećoj veb lokaciji za ovu knjigu: http://www.awl.com/kurose-ross.