



Zoran Bosnić in Mojca Ciglarič

Računalniške komunikacije

učbenik, delovna verzija v0.49

Ljubljana, 2023

Zahvaljujeva se sodelavcem Laboratorija za računalniške komunikacije in Laboratorija za kognitivno modeliranje, ki so bili pri nastajanju knjige nepogrešljivi in izkazujejo veliko predanost pri soustvarjanju predmeta. Hvala tudi prvotnima recenzentoma knjige, Andreju Brodniku in Patriciu Buliću, ter ilustratorjem prve izdaje, Tanji Malić in Mitji Puzigaća.

Kazalo

1 Zgodba o omrežjih	1
1.1 Splošno o omrežjih	2
1.1.1 Arhitektura Interneta	2
1.1.2 Dostop do omrežja	3
1.2 Komunikacija v omrežjih	6
1.2.1 Komunikacijski protokoli	6
1.2.2 Enkapsulacija	10
1.2.3 Protokolni sklad	11
1.2.4 Multipleksiranje protokolarnih zapisov	12
1.3 Referenčni modeli plasti	13
1.3.1 Fizična in povezavna plast	14
1.3.2 Omrežna plast	15
1.3.3 Transportna plast	17
1.3.4 Aplikacijska plast	18
2 Fizična plast	21
2.1 Naloge fizične plasti	21
2.2 Osnovni pojmi na fizični plasti	22
2.3 O elektromagnetnih signalih	23
2.4 Digitalno in analogno	26
2.5 Neželeni električni pojavi pri prenosu	28
2.6 Prenosni mediji	29
2.7 Kodiranje	33
2.8 Sklep	37
3 Povezavna plast	39
3.1 Storitve povezavne plasti	39
3.1.1 Okvirjanje datagramov	40
3.1.2 Zaznavanje in popravljanje napak	40
3.1.3 Dostop do medija	44
3.1.4 Ostale storitve povezavne plasti	53
3.2 Pomembni protokoli na povezavni plasti	57

3.2.1	Ethernet	57
3.2.2	WiFi – IEEE 802.11	65
3.2.3	Drugi protokoli iz družine IEEE 802	68
3.2.4	Point-to-point protocol (PPP)	70
4	Omrežna plast	73
4.1	Vrste omrežij	74
4.1.1	Povezavna omrežja	74
4.1.2	Nepovezavna omrežja	75
4.2	Storitve omrežne plasti	77
4.3	Protokol IPv4	79
4.3.1	Fragmentacija	79
4.3.2	Naslavljanje IPv4	83
4.3.3	Hierarhija pri naslavljjanju	85
4.3.4	Prevajanje omrežnih naslovov	87
4.4	Protokol IPv6	88
4.4.1	Naslavljanje IPv6	91
4.4.2	Prehodni mehanizmi z IPv4 na IPv6	93
4.5	Usmerjanje	95
4.5.1	Usmerjanje na podlagi stanja povezav	97
4.5.2	Usmerjanje z vektorjem razdalj	98
4.6	Varnost na omrežni plasti	99
5	Transportna plast	103
5.1	Storitve transportne plasti	103
5.2	Komunikacija med aplikacijo in omrežjem	104
5.3	Protokol UDP	105
5.4	Zanesljiva dostava	107
5.4.1	Pošiljanje in prejemanje podatkov	108
5.4.2	Potrjevanje in ponovno pošiljanje	108
5.4.3	Obvladovanje izgube segmentov	110
5.4.4	Neposredno in posredno potrjevanje	112
5.4.5	Tekoče pošiljanje	114
5.5	Protokol TCP	117
5.5.1	Vzpostavitev povezave in številčenje segmentov	119
5.5.2	Rušenje povezave	122
5.5.3	Stanja protokola TCP	122
5.5.4	Nastavitev časovne kontrole	123
5.5.5	Potrjevanje TCP	125
5.5.6	Nadzor pretoka	126
5.5.7	Nadzor zasičenja	126
5.6	Pravičnost TCP in UDP	129

5.6.1	Pravičnost TCP	130
5.6.2	Pravičnost UDP	131
6	Aplikacijska plast	133
6.1	Protokol HTTP	135
6.1.1	Oblike sporočil	136
6.1.2	Vrste povezav	138
6.1.3	Uporaba piškotkov	140
6.1.4	Uporaba spletnih medstrežnikov	142
6.2	Protokol FTP	143
6.3	Protokoli za elektronsko pošto	146
6.3.1	Protokol SMTP	147
6.3.2	Protokol POP	149
6.4	Imenska storitev DNS	151
6.4.1	Predpomnjenje zapisov DNS	153
6.4.2	Zapisi virov DNS	153
6.4.3	Protokol DNS	154
6.5	Protokoli P2P	156
6.5.1	BitTorrent	158
6.5.2	Skype	159
7	Kriptografija in omrežna varnost	161
7.1	Elementi omrežne varnosti	162
7.2	Preprosti kriptografski pristopi	164
7.2.1	Substitucijske metode	165
7.2.2	Transpozicijske metode	167
7.3	Data Encryption Standard (DES)	169
7.3.1	Trojni DES	172
7.3.2	Advanced Encryption Standard (AES)	173
7.3.3	Verižno kriptiranje blokov	174
7.3.4	Varnost bločnih kriptosistemov	176
7.4	Kriptografija z javnimi ključi	177
7.5	Pomožni kriptografski postopki	180
7.5.1	Zgoščevalne funkcije	180
7.5.2	Generatorji naključnih števil in generatorji praštevil	182
7.6	Avtentikacija pošiljatelja in prejemnika	183
7.6.1	Vzpostavljanje skupnega ključa: protokol Diffie-Hellman	185
7.6.2	Avtentikacijski protokoli	187
7.7	Integriteta komunikacije	195
7.7.1	Pripenjanje zgoščene vrednosti	195
7.7.2	Zgoščanje z avtentikacijskim ključem	195
7.7.3	Elektronski podpis	196

Predgovor

Spoštovane študentke in študenti!

Pred vami je delovna verzija tretje izdaja učbenika za predmet Računalniške komunikacije, ki je prvič našla pot v digitalni svet. Po devetih letih od pričetka pisanja učbenika, je karantena ob razglašeni epidemiji prišla prav za dokončanje učbenika in s tem izpolnitev obljud, danim sebi in študentom prejšnjih generacij. Za razliko od prvih dveh izdaj ta učbenik ni več zbirka poglavij domačih in tujih avtorjev, temveč delo učiteljev naše fakultete.

Vsebina knjige obsega opis arhitekture današnjih komunikacijskih sistemov in principov njihovega delovanja. Čeprav se vsebina bolj osredotoča na današnji Internet, pa so principi delovanja komunikacijskih sistemov splošni in prenosljivi tudi na druga področja računalništva in informatike. Z rastjo Interneta postaja računalniška komunikacija nepogrešljiv vezni element med tehnologijami, ljudmi in celinami. Nekoč nepremostljive geografske ovire so danes premostljive le z nekaj kliki na miško, računalniška komunikacija pa nas obdaja vsepo vsod v vsakdanjem življenju: pri uporabi mobilnih telefonov, deskanju po spletu, klepetanju, igranju iger, gledanju TV programov, uporabi elektronske pošte in druge.

Avtorja knjige vam želiva veliko uspehov pri osvajanju novih znanj in učenju nove snovi.

Zoran Bosnić in Mojca Ciglarič
Ljubljana, maj 2020

1 Zgodba o omrežjih

V zgodnjih devetdesetih letih je Bill Gates, prvi človek Microsoftovega imperija, jedrnato podal javno izjavo, da njegovo podjetje ni zainteresirano v nič, kar je povezano z Internetom, ker je ta samo modna muha in nima dolgoročne perspektive. Že v roku nekaj let se je izkazalo, da se ni mogel bolj zmotiti. Rastoče število računalnikov in večja dostopnost tehnologije sta povečevala potrebo po povezovanju sistemov in računalniški komunikaciji in tako smo postali priča rojevanju krajevnih omrežij, ki so se začela povezovati v omrežje vseh teh omrežij – *medmrežje* ali *Internet*.

Današnja omrežja niso namenjena samo osebnim računalnikom, temveč tudi mnogim drugim napravam, namen le-teh pa je medsebojna izmenjava podatkov. Med drugim so to tudi telefoni, mobilni računalniki, navigacijske naprave, sateliti, radijski sprejemni, televizorji, igrače, senzorji, kamere, nepričakovano pa lahko tudi gospodinjske naprave, kot so hladilniki in toasterji! Že s seznama naprav vidimo, da je uporabnost računalniških komunikacij rastočega pomena. Poleg spletnega brskanja za zabavo največje svetovno omrežje Internet danes poenostavlja življenje s preprostim dostopom do znanja, elektronskim poslovanjem, skupinskim sodelovanjem pri delu (angl. *crowdsourcing*), sodelovanjem na forumih, socialnim mreženjem, spletnim oglaševanjem, spletno prodajo in še z mnogimi drugimi storitvami, ki jih poznamo danes.

Študent ali morebitni računalniški radovednež, ki ravnokar drži v rokah knjigo s poglavji o računalniških komunikacijah, bi se v tej točki lahko vprašal, zakaj sploh govoriti o računalniških komunikacijah, če pa je tako zelo preprosto povezati računalnik z omrežno vtičnico in odpreti priljubljeni brskalnik ter začeti "surfati". Odgovora na to vprašanje sta dva. Prvi, krajši, je ta, da *uporaba spletja*, ki predstavlja le majhen fragment iz poglavij o računalniških komunikacijah, ni enaka *razumevanju* tega, kako splet deluje. Drugi, pomembnejši, je ta, da se je s spoznavanjem omrežnih tehnologij in pojmov/kratik kot so TCP, IP, NAT, SSL, RSA itd. moč naučiti splošnih konceptov in postopkov reševanja splošnih problemov v računalništvu. Pojmi, kot so zanesljivost, iskanje poti, naslavljjanje in deljena uporaba virov predstavljajo pomembno osnovo za raziskovanje in razvoj na področju računalniških komunikacij, prisotni pa so tudi na drugih področjih računalništva. Z razumevanjem delovanja računalniških komunikacij želimo bralcu na tem področju dati širino in poleg (upamo) potešitve radovednosti, omogo-

2 Poglavlje 1 Zgodba o omrežjih

čiti, da pridobi znanja, ki jih bo lahko uporabil tudi na področjih programiranja, kriptografije, umetne inteligence in drugod.

Z drugimi besedami, tudi če kakšen novodobni Gates II spregleda porast revolucionarne tehnologije, kot je to storil njegov predhodnik, podaja knjiga na primeru Interneta snov, ki je uporabna za implementacijo tehnologij, o katerih danes morda niti še ne sanjamo.

1.1 Splošno o omrežjih

Že v uvodu smo srečali že kar nekaj novih pojmov: **omrežje, komunikacija, sistem**. Začnimo torej na začetku in razložimo, kaj pomeni pojem *omrežje*. Poleg slovnične definicije, da je samostalnik srednjega spola, ga lahko v računalniškem kontekstu definiramo dvoplastno:

- Če nanj pogledamo s **fizičnega zornega kota**, lahko na omrežje gledamo kot na *vir* (torej množico naprav, priprav in drugih tehničnih stvari), ki omogoča, da različne naprave, ki smo jih našeli v prejšnjem razdelku, med seboj povežemo. Primeri sestavnih delov omrežij, ki jih bomo spoznali s tega vidika, so kabli, brezžične povezave, stikala in usmerjevalniki.
- Če nanj pogledamo s **storitvenega zornega kota**, lahko na omrežje gledamo kot *sredstvo*, ki nam omogoča izvajati storitve. O kakšnih *storitvah* govorimo? Spomnimo se prejšnje definicije, ki pravi, da je omrežje sestavljeno iz računalnikov in žic. Hitro lahko ugotovimo, da je ta fizična definicija sicer pravilna, vendar pa ne omenja *namena*, zakaj si dajemo s povezovanjem naprav sploh opravka. Naša druga, to je storitvena definicija, poudarja ravno manjkajoče, in sicer da je omrežje aplikacijska infrastruktura, ki izvaja storitve, kot so splet, elektronska pošta, deljenje datotek, elektronsko poslovanje in druge.

Ker se bomo učenja omrežij lotili od spodaj navzgor (od kablov in vijakov do omrežnih aplikacij, ki jih danes poznamo in uporabljam), se bomo tudi spoznavanja elementov omrežij lotili najprej skozi oči naše fizične definicije, ki ji bo sledila obravnava po storitveni definiciji.

1.1.1 Arhitektura Interneta

Z arhitekturnega vidika lahko omrežja razdelimo na tri bistvene komponente. Za njihovo ponazoritev si poglejmo sliko 1.1, ki jo tvorijo:

1. **Končni sistemi:** to so vse naprave, kot so namizni računalniki, strežniki, igralne konzole, televizorji, mobilni telefoni, avtomobili, spletne kamere in ostali – torej naprave, ki za svoje delovanje od omrežja zahtevajo povezljivost in so razlog, zakaj omrežje sploh obstaja. Končnim sistemom pravimo

tudi **gostitelji** (angl. *host*). Ne glede na to, ali se končni sistemi nahajajo doma ali v večjem podjetju, lahko iz slike tudi razberemo, da se lahko le-ti med seboj povezujejo v manjša krajevna omrežja (angl. *LAN - Local Area Network*)¹, ta pa se lahko med seboj povezujejo v večja omrežja. Lokalna omrežja so lahko na ta način del večjih omrežij ponudnikov omrežnih storitev, ta pa so del še večjih ponudnikov. Tak način povezovanja tvori veliko omrežje vseh omrežij, ki ga imenujemo medmrežje (angl. *internet*, z malo začetnico). Današnje najbolj razširjeno svetovno medmrežje pa imenujemo *Internet* (z veliko začetnico).

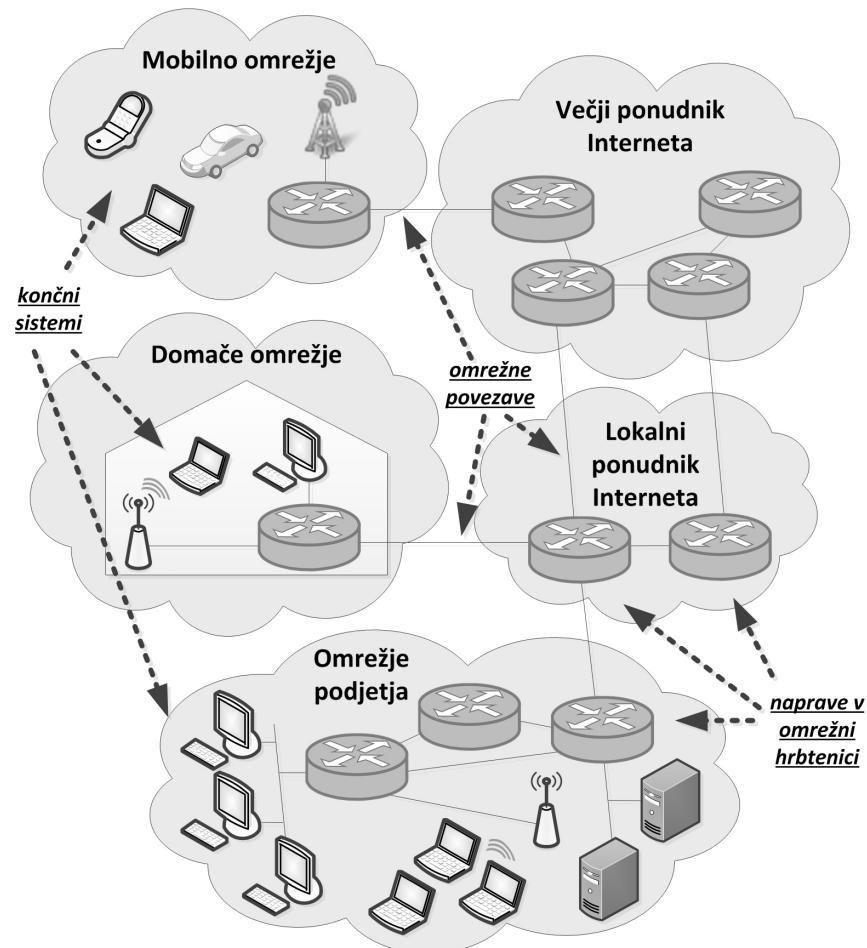
2. **Omrežna hrbtenica:** je omrežje naprav, ki povezujejo manjša omrežja v večje skupno omrežje in na ta način posredno zagotavljajo povezljivost med končnimi sistemi iz različnih omrežij. Našo pozornost bomo posebej posvetili t.i. *paketnim* ali *datagramskim* omrežjem, kot je tudi omrežje Internet. Na primeru teh omrežij bomo spoznali, kako naprave, kot so stikala (angl. *switch*) in usmerjevalniki (angl. *router*), zagotavljajo povezljivost omrežnih sistemov in skrbijo za ustrezeno usmerjanje podatkov med njimi.
3. **Povezave** v omrežni hrbtenici so lahko iz različnih fizičnih medijev: bakenih žic, optičnih kablov ali brezžičnih povezav, ki delujejo na osnovi elektromagnetnega valovanja. Namenjene so povezavi posameznih naprav v omrežni hrbtenici ali priključitvi končnih sistemov na omrežje.

1.1.2 Dostop do omrežja

Pri pregledu komponent omrežij smo omenili, da se končni sistemi priklapljam na omrežno hrbtenico preko različnih povezav do njihovih ponudnikov storitev (za Internet so to ponudniki internet storitev, angl. *ISP, Internet Service Provider*), ki so v splošnem lahko lokalna telekomunikacijska podjetja, univerze ali javni ponudniki storitev (npr. brezžičnega dostopa na letališčih in v hotelih). Ponudnik storitev je obenem tisti, ki opredeli, na kakšen način bodo končni sistemi lahko z njim vzpostavili povezavo in se vključili v izmenjavo podatkov v omrežju. Poglejmo si podrobneje nekaj najpogostejših načinov dostopa do omrežij, ki so ilustrirani na slikah 1.2 in 1.3:

1. **Modemski dostop:** danes zastareli, a v devetdesetih letih eden najbolj razširjenih načinov dostopa do Interneta. Uporabniki tega dostopa za prikllop na omrežje potrebujejo napravo, imenovano *modem*, ki jo na eni strani priključijo na svoj računalnik, na drugi strani pa (najpogosteje) na analogno telefonsko linijo. Ob vzpostavitvi povezave modem pokliče telefonsko številko ponudnika storitev (zato tej vrsti dostopa pravimo tudi *klicni dostop*) in

¹Lokalna omrežja so, glede na prostorsko razprostranjost, prvi tip omrežja, ki smo ga do sedaj omenili. Drugi pomembni so še MAN (angl. *Metropolitan Area Network*), PAN (angl. *Personal Area Network*) in WAN (angl. *Wide Area Network*).



Slika 1.1 Komponente omrežij: končni sistemi, omrežna hrbtenica in omrežne povezave

prične z izmenjavo podatkov, ki so kodirani z zvokom. Neprijetni slabosti tega dostopa sta njegova počasnost (največja hitrost prenosa znaša 56 kbps²) in zasedenost telefonske linije v času dostopa.

2. **Digitalna naročniška linija** (angl. *DSL, Digital Subscriber Line*): je danes najbolj razširjen način dostopa, ki v razvitih državah predstavlja 80% delež domačih dostopov do omrežij. Gre za način dostopa, ki je običajno vezan na obstoječo digitalno telefonsko linijo, ki pa se jo s posebnim napravo izkoristi za *multipleksiranje* signala – to pomeni, da isto linijo razdelimo v več frekvenčnih pasov, tako da lahko po njej teče več vzporednih komunikacij. Klasična telefonska linija se običajno deli v tri frekvenčne pasove³, od

²kbps - angl. *kilobits per second*, kilobit na sekundo

³Za primerjavo povejmo, da je na podoben način deljen tudi analogni televizijski signal, ki na različnih frekvencah prenaša različne TV programe.

katerih se pas v razponu 0-4 kHz uporablja za običajni prenos govora, pas v razponu 4 - 50 kHz za prenos podatkov navzven (angl. *upstream*) in pas v razponu 50 kHz - 1 MHz za prenos podatkov navznoter (angl. *downstream*). Kadar ponudnik dostopa DSL, ki je običajno kar lokalno podjetje za telefonijo, ponudi paket, kjer se hitrosti prenosa navzven in navznoter razlikujeta, pravimo, da gre za asimetrični DSL oziroma ADSL. Najvišje hitrosti pri dostopu ADSL segajo do 2.5 Mbps navzven in 24 Mbps navznoter.

3. **Kabelski dostop** (angl. *cable access*): podobno kot podjetja za telefonijo zagotavljajo dostop DSL, lahko dostop do Interneta zagotavljajo tudi kabelski operaterji. Ti namesto DSL modema namestijo podobno napravo, imenovano *kabelski modem*, ki s frekvenčno razdelitvijo linije obenem predstavlja tudi razdelilec med televizijskim in podatkovnim signalom. Dodatna razlika med dostopom DSL in kabelskim dostopom je tudi ta, da so lokalni kabelski modemi priključeni z bakrenim kablom na lokalno omrežno vozlišče, ki ga kabelski operater postavi kot centralo, namenjeno priklopu 500 - 5000 uporabnikov v krajevni bližini. Ker si veliko število uporabnikov deli isto komunikacijsko vozlišče (centralo), se lahko pojavijo težave, kadar je večje število uporabnikov hkrati aktivnih, saj zaradi delitve komunikacijskega medija hitrost prenosa pade. Pri kabelskem dostopu segajo najvišje hitrosti navzgor do 30.7 Mbps in navzdol do 42.8 Mbps.
4. **Optika do doma** (angl. *FTTH, Fiber to the home*): sodoben, cenovno dražji način povezave, ki pa postaja v zadnjih letih vse bolj razširjen in dostopen. Pri tem dostopu je optična povezava napeljana od ponudnika storitev vse do razdelilca (centrale), na katero je priključenih do 100 uporabnikov, ravno tako preko optičnih povezav. Čeprav najvišje hitrosti dostopa segajo do nekaj Gbps, so običajne ponujene hitrosti nižje, najpogosteje uporabljeni je 20 Mbps.
5. **Ethernet**: večja podjetja in univerze imajo običajno zagotovljen bolj neposreden dostop do interneta od sedaj opisanih, saj je usmerjevalnik v njihovem omrežju lahko priključen neposredno na enega usmerjevalnikov v hrbtenici Interneta. Uporabniki v omrežju teh podjetij in univerz lahko na ta način dostopajo do Interneta že kar s priklopom v lokalno omrežje, kjer se najbolj pogosta tehnologija lokalnih omrežij imenuje Ethernet. V Ethernetu so lokalni uporabniki običajno z večžilnim bakrenim kablom (kabel UTP, angl. *Unshielded Twisted Pair*) priključeni na lokalno napravo, imenovano stikalo, ki zagotavlja hitrost prenosa od 100 Mbps do 10 Gbps.
6. **WiFi**: Alternativa povezovanju z bakrenim kablom je uporaba brezžičnega lokalnega omrežja, ki je narejeno tako, da naprava, imenovana *dostopna točka* (angl. *access point*), omogoča povezavo med uporabniki, ki se s svojimi brezžičnimi omrežnimi vmesniki povezujejo nanjo, in najpogosteje omrežjem Ethernet, na katerega je dostopna točka fizično priključena. Za

6 Poglavlje 1 Zgodba o omrežjih

uspešno rabo signala WiFi se morajo uporabniki nahajati v razdalji do največ nekaj deset metrov od dostopne točke. Komunikacija v brezžičnem omrežju običajno poteka na podlagi standarda IEEE 802.11, ki je drugačna tehnologija kot Ethernet in zagotavlja hitrosti prenosa do 1 Gbps (standard IEEE 802.11ac, podatek iz leta 2015).

7. **3G in 4G:** Zaradi porasta števila mobilnih naprav se povečuje tudi potreba po dostopu do Interneta, multimedijskih vsebin, spletu, elektronske pošte itd. tudi s teh naprav. Za ta namen lahko naprave izkoristijo *bazne postaje* (t.j. centrale mobilnih operaterjev), preko katerih lahko namesto govora prenašajo tudi podatke. Takšne brezžične povezave, ki delujejo tudi na razdalje do nekaj deset kilometrov, imenujemo brezžične povezave tretje (3G) in četrte generacije (4G). Dosegajo najvišje hitrosti okoli 1 Mbps (3G) oziroma 10 Mbps (4G).

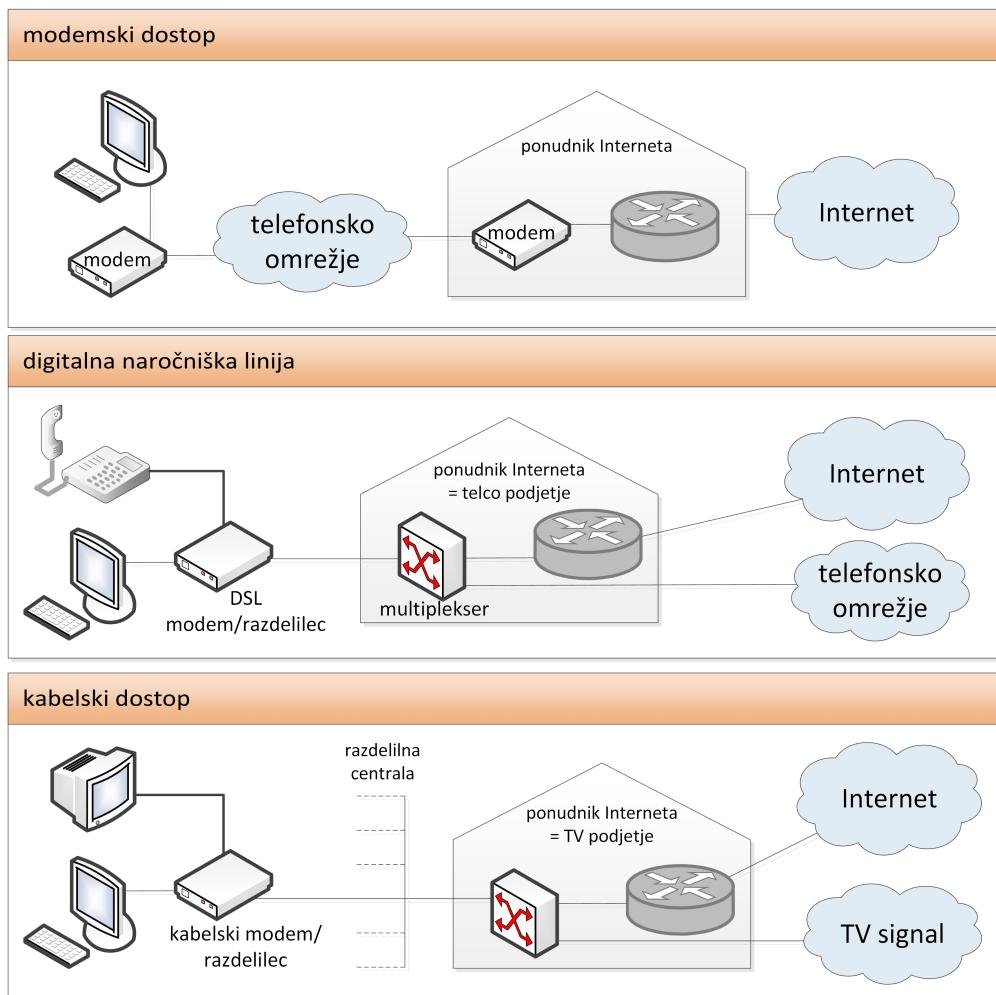
Z arhitekturno razdelitvijo komponent omrežij in s preletom načinov priklopa uporabnikov (končnih sistemov) v hrbtenico omrežja, smo naredili pregled nad fizičnimi komponentami, ki tvorijo omrežja. Sedaj je čas, da naredimo korak dlje in iz tlorisa orišemo potek komunikacije v omrežjih.

1.2 Komunikacija v omrežjih

Kot smo že uspeli razbrati, opravljajo omrežni sistemi vrsto zahtevnih nalog. Da lahko omrežne aplikacije nemoteno delujejo, morajo omrežja poskrbeti za celo vrsto nalog, kot so iskanje poti skozi omrežno hrbtenico, vzpostavljanje in prekinitve povezav, prenos podatkov poljubne dolžine, skrb za zanesljivo dostavo podatkov, optimizacija hitrosti prenosa in varovanje podatkov med prenosom. Poleg zagotavljanja teh storitev aplikacijam morajo omrežja tudi omogočati preprosto priključevanje in zapuščanje uporabnikov. Na osnovi povedanega lahko torej rečemo, da morajo omrežja biti *visoko modularna*. To modularnost omogočata večplastna zasnova omrežnih sistemov, ki jo bomo podrobno spoznali v nadaljevanju, in širok nabor protokolov, ki lahko izvajajo različne naloge (omrežne storitve).

1.2.1 Komunikacijski protokoli

Modularnost v omrežjih dosežemo tako, da posamezne naloge, vezane na prenos podatkov, razdelimo med manjše skrbnike teh nalog, ki jih imenujemo **protokoli**. En protokol bo tako lahko npr. skrbel za iskanje poti skozi omrežje, drugi za varovanje podatkov med prenosom, tretji pa za prenos podatkov do sosednjega vozlišča v omrežju. Protokol lahko definiramo kot seznam pravil za izvedbo komunikacije in je lahko vezan na določen standard. Nanj lahko gledamo tudi kot na jezik za sporazumevanje, ki ga morata govoriti udeleženca v

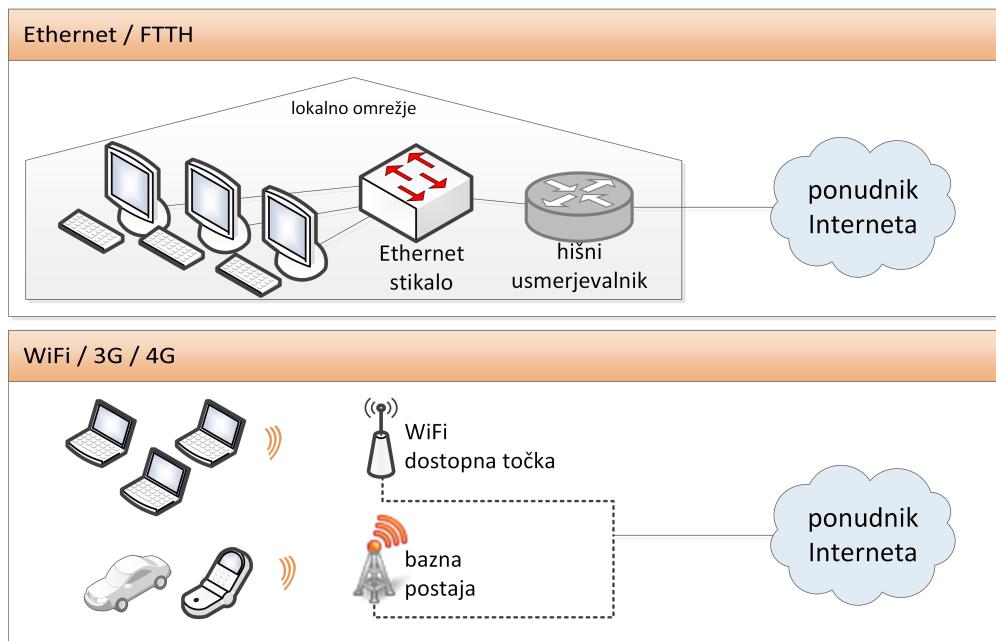


Slika 1.2 Načini dostopa do omrežja: modemski dostop, digitalna naročniška linija in kabelski dostop

komunikaciji, če želita, da se razumeta pri opravljanju določene naloge, povezane s prenosom podatkov. Zato morata oba udeleženca v komunikaciji biti sposobna razumeti in izvajati isti protokol, na osnovi katerega si izmenjujeta **protokolarna sporočila**.

V komunikacijskih sistemih sta lahko dva protokola med seboj odvisna na ta način, da uporablja en protokol za izvedbo svoje naloge storitve drugega protokola. Pri takšni odvisnosti veljata naslednji pravili, ki ju ponazarjamо tudi s sliko 1.4:

1. Če vozlišče A komunicira z vozliščem B z uporabo protokola P, se protokolarna sporočila med A in B izmenjujejo v obliki, ki jo določa protokol P ne glede na morebitne odvisnosti izvajanja protokola P od drugih protokolov.



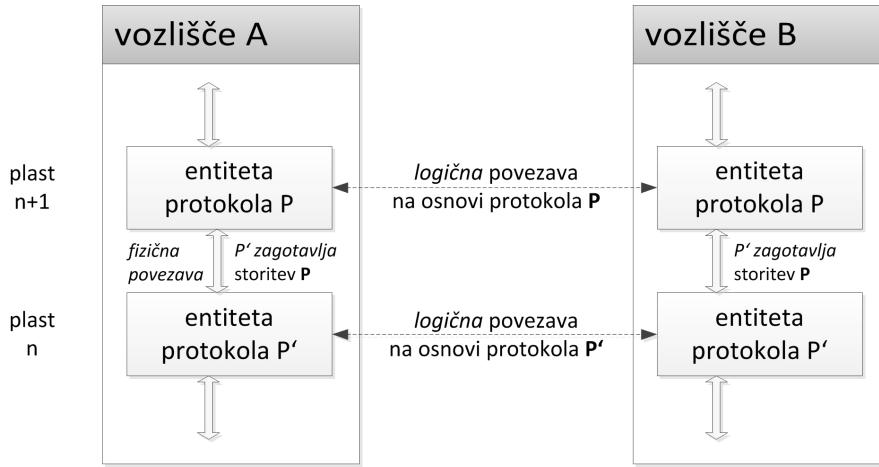
Slika 1.3 Načini dostopa do omrežja: Ethernet, FTTH, WiFi, 3G/4G

Pravimo, da poteka med vozliščema **logična** ali **horizontalna povezava** pri sporazumevanju s protokolom P.

2. Če vozlišče A komunicira z vozliščem B z uporabo protokola P, ki za izvedbo svoje naloge potrebuje protokol P', pravimo, da se protokol P' izvaja na *nižji plasti* kot protokol P in da P' *zagotavlja svojo storitev* protokolu P. Med vozliščema A in B obstajata tedaj dve logični povezavi: pri sporazumevanju s protokolom P in pri sporazumevanju s protokolom P'. Med protokoloma P in P' pa obstaja **fizična** ali **vertikalna povezava**, saj se izvajata znotraj posameznih vozlišč, med njima pa velja hierarhična odvisnost.

Iz zapisanih pravil lahko izluščimo, da so protokoli razvrščeni v plasti, ki jih je v splošnem lahko tudi več. Vsako vozlišče hrani svojo *protokolarno entiteto*. Nižje plasti protokolov imajo nalogu izvajati storitve na zahtevo višjih plasti. Za vse plasti velja, da so med seboj povezane *fizično* ali *vertikalno*, saj povezave med plastmi tečejo v istem omrežnem vozlišču. Medtem pa so protokolarne entitete med vozlišči povezane *logično* ali *horizontalno*, saj med seboj vedno komunicira le par na istoležni plasti. Za logično komunikacijo protokolarne entitete ne uporabljajo fizične povezave med vozlišči kabla (zato se tudi povezavi reče, da je le *logična*), temveč lahko komunicirajo posredno preko protokolarne entitete na nižji plasti, od katere zahtevajo izvedbo storitve za del svoje naloge.

V tej točki smo z razlago prišli že tako daleč, da bi bil čas, da za primer omenimo nekaj pravih protokolov, ki se uporabljajo v Internetu, in nakažemo

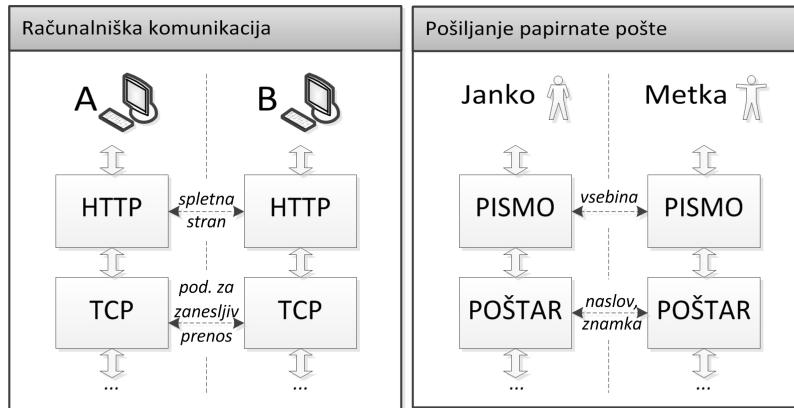


Slika 1.4 Komunikacija med protokoli na istoležnih plasteh

njihovo povezanost v smislu odvisnosti delovanja. Ena najbolj razširjenih aplikacij je zagotovo spletni brskalnik, ki uporablja protokol *HyperText Transfer Protocol* (HTTP), ki ga bomo podrobneje spoznali v poglavju 6. Ker je potrebno zagotovilo, da pri prenosu spletnih strani ne bo prišlo do okvar zaradi motenj v komunikaciji, protokol HTTP uporabi protokol nižje plasti, in sicer protokol TCP (*Transmission Control Protocol*). Slednji skrbi za pravilno urejenost prenesene vsebine in brezizgubnost podatkov. HTTP uporabi protokol TCP tako, da podatke v sporočilu HTTP opremi z dodatnimi podatki, ki jih uporablja protokol TCP, in vse skupaj predstavi kot razširjen zapis podatkov, ki ga mora pošiljatelj dostaviti prejemniku.

Ta postopek je zelo podoben temu, kako v vsakdanjem življenu poteka pošiljanje navadne (papirnate) pošte, kar je prikazano na sliki 1.5. Če želi Janko poslati ljubezensko pismo Metki (ljubezensko pismo je analogno našemu sporočilu HTTP), mora svoje pismo vstaviti v kuverto ter uporabiti poštno storitev, ki bo pismo dostavila. Za to, da poštar ve, kam odnesti pismo, mora biti Jankova kuverta opremljena z ustreznimi podatki, vsaj z znamko in naslovom prejemnika. Na enak način kot je pošta storitev, ki jo uporabi Janko, je v našem prejšnjem primeru tudi protokol TCP storitev, ki jo uporabi protokol HTTP, da lahko na zanesljiv način izvede pošiljanje svojega sporočila. Na podoben način lahko protokol TCP in poštar zahtevata storitve od protokola, ki je na morebitni še nižji plasti (zato slika prikazuje puščice, ki ju povezujejo z nepričazanimi nižjimi plasti). Kmalu bomo razkrili, da v Internetu TCP tudi dejansko zahteva storitve protokola IP, poštar med Jankom in Metko pa lahko morda uporabi storitve mestnega potniškega prometa.

V našem primeru je pomembno, da poštarja (razen, če je ljubosumni zlobnež, ki je tudi zaljubljen v Metko, za kar predpostavimo, da ni), ne zanima, kaj Janko pošilja Metki v kuverti. Zanj so namreč pomembni samo tisti podatki, ki so



Slika 1.5 Primerjava komunikacijskega modela z modelom pošiljanja navadne (papirnate) pošte

zapisani na kuverti, medtem ko branje vsebine pisma ni del njegovega poklica in če ne pozna Janka in Metke, pisma tudi ne bi razumel. Na enak način delujeta tudi HTTP in TCP, saj TCP ne ve, kakšno sporočilo mora dostaviti prejemniku (v našem primeru namreč protokol TCP nudi storitev prenosa protokolu HTTP, v splošnem pa bi lahko nudil prenos tudi kakemu drugemu protokolu, kot so npr. SMTP, POP3, IMAP, FTP). Za protokol TCP na strani prejemnika so pomembni samo tisti podatki, ki jih je protokol TCP na strani pošiljatelja dodal originalnim podatkom, torej vsebini sporočila HTTP. In istočasno bo protokol HTTP na prejemnikovi strani zanimal samo tisti del sporočila, ki ga pošilja protokol HTTP s strani pošiljatelja, brez dodatkov protokola TCP. Zato pravimo, da ista protokola pri pošiljatelju in prejemniku komunicirata kot **navidezni logični komunikacijski par**, saj se komunikacija med njima odvija neodvisno od tega, kateri protokol na nižji plasti je bil uporabljen za dostavo protokolarnega sporočila.

1.2.2 Enkapsulacija

Vidimo, da sta kuvertiranje pisne pošte in odvisnost protokolov podobna postopka. Na področju računalniških komunikacij takšen postopek, kjer se sporočilu enega protokola dodajo kontrolni podatki drugega protokola ter se s tem prvotno sporočilo prevede v sporočilo drugega protokola, imenujemo **enkapsulacija**. Bolj strokovno lahko torej rečemo, da se sporočilo protokola HTTP enkapsulira v sporočilo protokola TCP tako, da se sporočilu HTTP doda nova **glava sporočila** (množica podatkov, s katero se sporočilo začne, nato pa ji sledi vsebina). Nato se takšni razširjeni podatki TCP (na način, ki ga bomo še opisali) pošljejo prejemniku. Pri njem se postopek odvije v obratno smer, saj se iz zapisa TCP podatki **dekapsulirajo** nazaj v sporočilo HTTP, ki ga prejemnik nato lahko uporabi.

Polovico naše zgodbe o pošiljanju zahtevka HTTP smo do sedaj uspeli zamolčati, zdaj pa je čas, da pojasnimo še njen preostanek. Namreč, podobno kot uporablja protokol HTTP za storitev zanesljive dostave protokol TCP, tudi TCP ne opravi celotne naloge sam. Za dostavo podatkov potrebuje TCP ključno storitev komuniciranja – poiskati mora pot skozi veliko omrežje od pošiljatelja do prejemnika. Za ta namen TCP uporablja protokol IP (*Internet Protocol*), ki omogoča podatke ustreznno usmeriti po Internetu, da prepotujejo pot od vozlišča do vozlišča, z mnogimi postanki. Ta pot je lahko zelo dolga in se začne pri domačem računalniku, kjer paketi protokola IP opravijo prvi delček poti že z uporabo naše morebitne domače brezžične povezave (torej protokol 802.11), s čimer so podatki naredili svoj prvi korak proti Internetu.

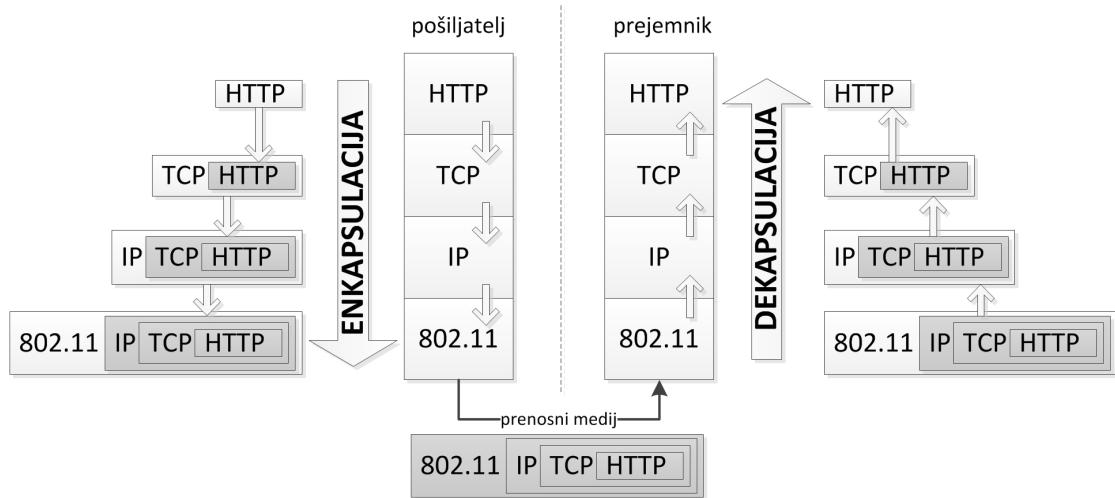
Enako kot se sporočilo HTTP enkapsulira v sporočilo protokola TCP, se sporočilo protokola TCP enkapsulira v paket protokola IP (protokol na plasti, ki je nižja od plasti s protokolom TCP), to pa v okvir protokola IEEE 802.11 (to je primer protokola plasti, ki leži pod plastjo s protokolom IP). Z vsakim od teh korakov prejšnje sporočilo dobiva novo glavo sporočila in z zaporedno enkapsulacijo postaja večkrat ovito, podobno kot ima čebula več plasti. Sporočila na najnižji plasti, ki se ukvarja z elektromagnetnim prenosom kodiranih bitov preko medija, ne enkapsuliramo več, ampak ga le prenesemo po prenosnem mediju na prejemnikovo stran. Tam pa se sporočilo dekapsulira v obratni smeri, da ga prejemnik lahko uporabi. Celoten opisani postopek, ki se odvije v našem primeru, prikazuje slika 1.6.

Do sedaj smo spoznali, kako so protokoli razvrščeni v plasti in ugotovili smo, da protokoli na nižjih plasteh zagotavljajo storitve protokolom na višjih plasteh. V naslednjem razdelku si poglejmo organizacijo podrobneje, poglejmo pa si tudi najbolj znana pristopa razvrščanja protokolov v plasti.

1.2.3 Protokolni sklad

V računalniških komunikacijah obstaja zelo veliko komunikacijskih protokolov, ki imajo med seboj tudi povsem različne naloge. Kot smo spoznali v prejšnjem razdelku, lahko med protokoli obstajajo odvisnosti v smislu, da eden protokol zagotavlja storitev drugemu. Takšne odvisnosti niso mogoče med vsemi poljubnimi kombinacijami protokolov, zato je protokole smiselno razdeliti v večplastni model, ki mu pravimo tudi **protokolni sklad**, s katerim definiramo namen posameznih protokolov, ki sodelujejo v komunikacijskem sistemu. Vsaka plast definira množico funkcij, ki so si po namenu ali delovanju sorodne. Takšna razdelitev je za računalničarje in snovalce komunikacijskih sistemov zelo koristna, ker:

- omogoča sistematično in logično razčleniti delovanje posameznih komponent sistema in s tem ponuja večje razumevanje njegovega delovanja,



Slika 1.6 Postopek enkapsulacije in dekapsulacije

- omogoča **modularno zasnovo sistemov** (po principu lego kock) ter neodvisnost in zamenljivost elementov v protokolnem skladu.

Omenjena modularnost sistema je še kako pomembna. Omogoča namreč, da lahko spremenimo le del programske opreme, ki je zadolžena za komunikacijo, če želimo komunikacijo izvesti na drugačen način. Navezujoc se na primer s slike 1.6 bi takšno modularnost lahko s pridom izkoristili, če bi komunikacijo namesto po brezžični povezavi (protokol IEEE 802.11) žeeli izvesti po bakreni povezavi (žici), kjer bi namesto IEEE 802.11 uporabili protokol Ethernet. Za ta namen nam ne bi bilo potrebno napisati popolnoma novega protokola HTTP, ki bi bil namenjen prenosu sporočil po bakrenih povezavah, temveč zadošča, da v protokolnem skladu zamenjamo le komponento na najnižji prikazani plasti.

1.2.4 Multipleksiranje protokolarnih zapisov

Modularnost omogoča, da lahko neko protokolarno sporočilo enkapsulira različne vrste protokolov. Tipičen primer tega je protokol IP, ki lahko kot podatek prenaša protokolarno sporočilo (ki ga imenujemo *segment*) protokola TCP ali pa protokolarno sporočilo (*datagram*) protokola UDP. Prejemnik mora na svoji strani torej vedeti, kaj je enkapsulirano v protokolarni enoti, ki jo je prejel, saj mora njenoutrajnost pravilno predati ustreznemu protokolu v nadaljnjo obdelavo. Če tako npr. v paketu IP prenašamo segment TCP, mora prejemnik vsebino (zaporedje bitov) tega paketa ob prejemu tudi interpretirati, kot da gre za segment TCP in ne kot morebitni datagram UDP. S tem namenom imajo protokolarni zapis posebno polje v glavi, ki ga imenujemo **ključ za demultipleksiranje**. Čeprav izraz zveni učeno, ne gre za nič drugega kot za oznako (številko) protokola, ki je

enkapsuliran v protokolarnem zapisu.

Za lažje razumevanje povežimo pojem demultipleksiranja z realnim primerom carinjenja navadne (papirnate) pošte. Kadar pošiljka (protokolarno sporočilo, ki ima v sebi enkapsulirano notranje sporočilo - vsebino pošiljke) prispe na carino, zanima carinike, kaj se nahaja v pošiljki. S tem namenom mora na pošiljki biti napisana kratka oznaka vsebine (ključ za demultipleksiranje). Na podlagi te oznake se carinik lahko odloči, kaj bo naredil s pošiljko: če piše "dokumenti", lahko pošiljko preda naslovniku brez carinjenja (prvi protokol); v primeru, da piše "hrana in pijača", pred pošiljko v pregled sanitarni inšpekciji (drugi protokol); v primeru, da piše "audio-video tehnika", pred pošiljko uradu za obračun carine (tretji protokol).

1.3 Referenčni modeli plasti

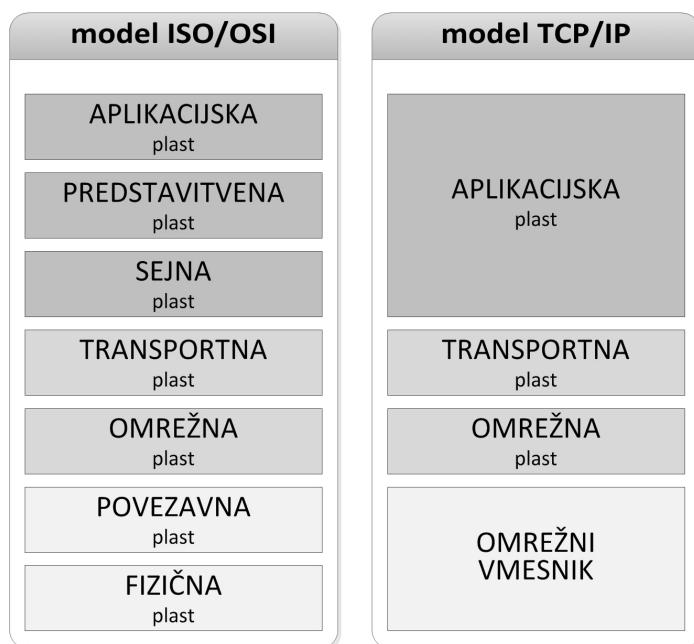
Danes najbolj znana modela, ki definirata funkcionalnosti komunikacijskih sistemov, sta **model ISO/OSI** in **model TCP/IP**. Njuna primerjava je prikazana na sliki 1.7. Kratica ISO v imenu prvega je akronim združenja⁴, ki je definiralo model, imenovan OSI (*Open Systems Interconnection*). Ta opredeljuje delitev funkcionalnosti komunikacijskega sistema sedem plasti, ki si od najvišje do najnižje plasti sledijo v naslednjem vrstnem redu: aplikacijska plast (angl. *application layer*), predstavitevna plast (angl. *presentation layer*), sejna plast (angl. *session layer*), transportna (ali prenosna) plast (angl. *transport layer*), omrežna plast (angl. *network layer*), povezavna plast (angl. *link layer*) in fizična plast (angl. *physical layer*).

V nadaljnjih poglavijih knjige se bomo podrobneje lotili obravnave vsake od naštetih plasti. Pri tem bomo spoznali, da je model ISO/OSI zelo natančen in razčlenjujoč, saj z vsako plastjo natančno opredeljuje storitev, ki naj bi jo implementirala vsaka omrežna aplikacija. Vendar pa v praksi ni vedno tako, saj so omrežne aplikacije lahko tudi dokaj preproste in ne potrebujejo vseh sedmih plasti funkcionalnosti. Iz navedenih razlogov je v praksi zato bolj zaživel **model TCP/IP**, ki sloni na praktičnih izkušnjah programerjev omrežnih aplikacij in ne toliko pretirano razčlenjenih smernicah, zapisanih v standardih in predstavljenih z modelom ISO/OSI.

Model TCP/IP zgradbo komunikacijskega sistema poenostavlja na naslednje štiri komponente: *fizično plast* (ki združuje fizično in povezavno plast modela ISO/OSI), *omrežno plast*, *transportno plast* in *aplikacijsko plast* (ta združuje aplikacijsko, predstavitevno in sejno plast modela ISO/OSI). Plasti modela TCP/IP, ki združujejo več plasti bolj obširnega modela, ne izpuščajo torej funkcionalnosti treh neomenjenih plasti modela ISO/OSI, temveč jih implementirajo skrivoma –

⁴International Organization for Standardization, mednarodno združenje več pravnih subjektov, zadolženih za definiranje komunikacijskih standardov.

znotraj aplikacijske plasti, na nekoliko manj standardiziran in zase bolj priročen način. Za model ISO/OSI zato tudi pravimo, da je dobro utemeljen v teoriji oziroma, da je **de-iure** (formalen, potren, dejanski uradni standard) model, saj natančno razčlenjuje zgradbo komunikacijskega sistema. Model TCP/IP, ki pa se je zaradi praktičnih razlogov bolje uveljavil, označujemo z izrazom **de-facto** (dejanski) model. Ker je Internet ustvarjalno okolje velikega števila programerjev, uporabnikov, heterogenih tehnologij in idej, si bomo v nadaljevanju podrobnejše pogledali plasti komunikacijskega modela TCP/IP in orisali njihove naloge.



Slika 1.7 Primerjava modelov OSI in TCP/IP

1.3.1 Fizična in povezavna plast

Najnižja plast modela TCP/IP, plast omrežnega vmesnika, združuje funkcije dveh plasti modela ISO/OSI: fizične in povezavne plasti. Najnižje naloge te plasti so bolj fizikalne/elektromagnetne kot računalniške narave, povezane pa so z vprašanjem, kako predstaviti elementaren računalniški podatek (torej bit oziroma vrednosti 0 in 1) z različnimi fizikalnimi količinami, kot so električna napetost, radijsko ali svetlobno valovanje. Bitje želimo predstaviti tako, da jih bo mogoče čim bolj učinkovito prenesti po fizičnih medijih, od katerih smo že spoznali bakreni medij, optično vlakno in zrak (medij za prenos brezžičnih signalov).

Ko ugotovimo, kako najbolj učinkovito prenesti vrednost bita po mediju, se nam lahko porodi naslednje smiselno vprašanje: Kako prenesti sporočilo, ki

je sestavljeno iz več bitov? Težav, ki jih srečamo pri iskanju odgovora na to vprašanje, je več. Najprej, če želimo prenesti več zaporednih sporočil, moramo prejemniku sporočila dati vedeti, kje je začetek in kje je konec vsakega od poslanih sporočil. Ker je prejemnikov našega sporočila lahko tudi več, je potrebno tudi označiti, komu so sporočila namenjena. Ker pa pri razbiranju sporočil lahko pride do napak, je koristno, da v naša sporočila vgradimo nekakšno *trdoživost, odpornost* proti napakam.

Za lažjo predstavo o problemu, ki ga opisujemo, si predstavljajmo dve indijanski plemeni, ki se sporazumevata s pomočjo dimnih signalov. Pleme, ki živi ob morju, sporoča plemenu v gorah, koliko ladij je ta dan plulo proti vzhodu in koliko ladij je plulo proti zahodu (pleme sporoča ta dva podatka nenehno in izmenično). Pleme izvaja to tako, da naredi toliko dimnih oblačkov, kolikor je bilo ladij na obzorju, začetek in konec vsakega sporočila pa označi z oblačkoma "posebne" oblike. Vendar pa lahko sporočila obmorskega plemena prebira tudi tretje pleme, ki na travniku pase ovce in pričakuje sporočila o vremenski napovedi. Da lahko obmorsko pleme nedvoumno določi, komu je sporočilo namenjeno, pripne na začetek še nekaj oblačkov, ki so tudi "posebne" oblike in predstavljajo oznako prejemnika (gorsko ali pastirsko pleme). Da ne bi kakšna ptica, ki bi preveč aktivno poletela skozi oblaček, pokvarila sporočila, naše obmorsko pleme opravi še eno nalogu: isto sporočilo pošlje dvakrat zapored, da zagotovi uspešen prejem tudi v primeru nepričakovanih situacij.

Vse našteto in še mnogo več so naloge povezavne plasti, ki pa ne skrbi za dimne signale, ampak za prenos bitov. Na povezavni plasti združimo zaporedje bitov v večjo enoto, imenovano **okvir** (angl. *frame*). Začetek in konec vsakega okvirja sta označena s posebnima zaporednjema bitov, imenujemo ju *glava* in *rep*, na podlagi teh prejemnik natančno ve, kje je začetek in kje konec okvirja. V okvirju sta zapisana tudi naslov pošiljatelja in prejemnika, dodatno pa so pripeti še posebni biti, ki se uporabljajo za zaznavanje in popravljanje morebitnih napak.

O še ostalih nalogah povezavne plasti bomo bolj podrobno spregovorili v poglavju 3. Do takrat pa navedimo samo nekaj protokolov povezavne plasti, ki smo jih že spoznali in ki skrbijo za okvirjanje bitov in za prenos po različnih medijih: Ethernet, 3G/4G, 802.11, DSL, kabelski internet in PPP. Naprave, s katerimi na povezavni plasti skrbimo za povezljivost vozlišč in prenos po posameznih omrežnih povezavah, imenujemo *omrežna stikala*.

1.3.2 Omrežna plast

S fizično in povezavno plastjo smo rešili problem prenosa zaporedij bitov (okvirja) preko ene same povezave. Kot eno povezavo pri tem mislimo neposredno povezavo med dvema napravama ali pa povezavo med napravama, ki se nahajata v istem krajevnem omrežju (LAN). Spomnimo se, da krajevno omrežje tvorijo naprave, ki so vzajemno povezane preko ene same povezave. Za povezovanje

naprav se uporabljajo naprave, imenovane *stikala*. Vemo pa, da je Internet prav-zaprav omrežje velikega števila povezanih manjših omrežij. In če želimo poslati podatke z enega konca Interneta na drugega, moramo imeti možnost iskanja ustrezne poti po več povezavah skozi vsa omrežja, ki vodijo do prejemnika.

Glavne naloge omrežne plasti so torej tri: izvajati usmerjanje (ustvarjanje poti skozi omrežje), izvajati naslavljjanje pošiljatelja in prejemnika na globalni ravni (ne samo znotraj lokalne povezave, temveč enolično v okviru celotnega širšega omrežja) in signalizirati o morebitnih napakah v omrežju.

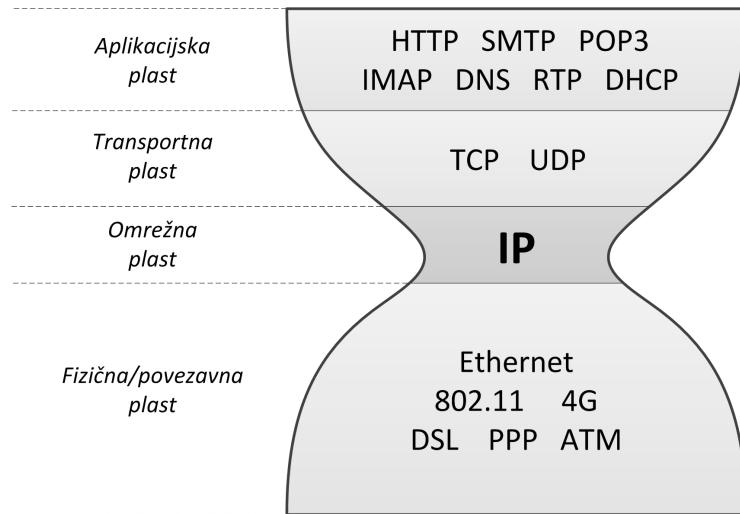
Za povezovanje večjega števila omrežij skrbijo naprave, imenovane *usmerjevalniki* (angl. *routers*). Večini od nas je cenejšo različico takšne naprave doma skoraj zagotovo priskrbel ponudnik interneta, domači usmerjevalnik pa skrbi, kot že povedano, za povezovanje našega domačega lokalnega omrežja z omrežjem internetnega ponudnika. Usmerjevalniki med seboj komunicirajo z uporabo različnih protokolov, s katerimi se obveščajo o tem, kateri usmerjevalnik vodi do katerega ciljnega omrežja in s tem tudi prejemnika.

Najbolj znan omrežni protokol in edini protokol, ki se v Internetu uporablja na omrežni plasti, je protokol IP (angl. *Internet Protocol*). To je protokol, katerega enote imenujemo **paketi** (angl. *packets*). Vsak paket IP je opremljen z naslovom pošiljatelja in prejemnika in dodatnimi kontrolnimi podatki, od katerih je odvisen način prenosa paketa skozi omrežje. Če pride pri prenosu do težav ali pa če potrebujemo podatke o stanju omrežja, je poleg protokola IP na razpolago tudi kontrolni protokol ICMP (angl. *Internet Control Message Protocol*), ki se uporablja za signalizacijo in pošiljanje obvestil o dogodkih in stanju omrežja (ta protokol uporablja tudi znani aplikaciji Ping in Traceroute).

Za protokol IP pravimo, da je datagramski (nepovezaven) protokol in da zagotavlja storitve dostave po najboljših zmožnostih (angl. *best-effort service*). To pomeni, da se lahko podatki pri prenosu tudi okvarijo, izgubijo, dospejo podvojeni ali v napačnem vrstnem redu. Zanesljivost pri dostavi je potrebno torej reševati z drugimi protokoli, na višjih plasteh komunikacijskega modela.

Ker je v omrežju Internet protokol IP edini protokol, ki deluje na omrežni plasti, predstavljajo njegove omejitve tudi omejitve celotnega protokolnega sklada, ki se uporablja v Internetu. Težave s protokolom IP izvirajo iz dveh nasprotujočih si ciljev: (1) narediti protokol dovolj splošen, da se lahko uporablja s poljubnimi transportnimi in povezavnimi protokoli ter (2) zagotoviti čim več storitev na omrežni plasti (bolj kot želimo narediti protokol splošen, manj specifičnih storitev lahko implementira). Glede na to, da je za zagotavljanje splošnosti nabor implementiranih storitev ostal majhen, morajo breme nadomeščanja teh storitev prevzemati protokoli na drugih plasteh ali pa aplikacije. Omejitev protokola IP velikokrat prikažemo kot ozko grlo (angl. *narrow waist*) na shemi v obliki peščene ure, kot je prikazana na sliki 1.8.

Za konec še navedimo, da se v Internetu trenutno vzporedno uporabljata dve verziji protokola IP: verzija 4 (IPv4) in verzija 6 (IPv6). Izkazalo se je, da



Slika 1.8 Protokol IP kot ozko grlo v protokolarnem skladu Interneta

je starejša, verzija 4, iz leta 1981, slabo predvidela porast števila uporabnikov Interneta in slabše zagotavlja prenos *posebnih* vrst podatkov, kot so multimedij-ske vsebine in podatki v realnem času. Zato je bila vzporedno razvita novejša verzija tega protokola, verzija 6, ki odpravlja pomanjkljivosti predhodnika (vmesna eksperimentalna verzija, IPv5, ni nikoli zaživela v praksi).

Omrežno plast si bomo podrobno ogledali v poglavju 4.

1.3.3 Transportna plast

Videli smo, da protokol IP ne skrbi za zanesljivost pri dostavi podatkov in da deluje po principu dostave po najboljših zmožnostih, kar v domačem jeziku pomeni: "Paket pride, če pride." Manjkajočo nalogo mora torej prevzeti nekdo drug, to pa je protokol transportne (ali pa kakšne še višje) plasti.

Med naloge, ki jih lahko ima transportni protokol, spada zagotavljanje transparentnega, učinkovitega in zanesljivega prenosa podatkov med končnima sistemoma. *Transparenten* pomeni, da transportni protokol aplikacijsko plast popolnoma razbremeni skrbi glede podrobnosti omrežne infrastrukture in jo prikaže kot črno škatlo, ki nudi storitev prenosa podatkov. *Učinkovit* in *zanesljiv* pomeni, da lahko transporten protokol zagotavlja tudi storitve zagotovljenega, varnega in celovitega prenosa.

Transportna protokola, ki se najpogosteje uporabljata v Internetu, sta TCP (angl. *Transmission Control Protocol*) in UDP (angl. *User Datagram Protocol*), med katerima so občutne razlike. Medtem ko je za prenos s protokolom TCP potrebno najprej vzpostaviti povezavo (pravimo, da je protokol *povezaven*), pro-

tokol UDP tega vzpostavitvenega postopka nima (je *nepovezaven*). Še bolj pomembno je dejstvo, da TCP dosledno skrbi za zanesljivo dostavo, protokol UDP pa je silno preprost in tega mehanizma nima. Posledično se TCP uporablja za prenos podatkov, kjer aplikacija potrebuje večja zagotovila pri prenosu.

Kljub temu to ne pomeni, da je protokol UDP neuporaben. Čeprav ne zagotavlja zanesljive dostave, ga uporabljamo v aplikacijah, pri katerih lahko toleriramo izgubo, nepravilen vrstni red in podvojene podatke na račun tega, da morajo podatki biti dostavljeni hitro in pravočasno. Takšne so npr. aplikacije, ki tečejo v stvarnem (realnem) času, kot so prenos zvoka, slike ali govora po omrežju (VoIP). Če za našo aplikacijo, ki je vezana na uporabo UDP, nezanesljiva dostava vendarle predstavlja problem, pa ga je potrebno rešiti – breme teh težav v primeru uporabe protokola UDP preložimo na aplikacijo samo.

Podrobnosti transportne plasti in mehanizme delovanja protokolov TCP in UDP si bomo natančno pogledali v poglavju 5.

1.3.4 Aplikacijska plast

Aplikacijska plast (angl. *application layer*) je plast, ki je najbližje uporabniku. To pomeni, da že kar "komunikacijski" del same uporabniške aplikacije uporablja aplikacijske protokole. Za začetek naštejmo nekaj aplikacijskih protokolov, ki so nam zagotovo znani iz vsakdanje rabe Interneta:

- **HTTP** (angl. *HyperText Transfer Protocol*): protokol za prenos spletnih strani (hiperteksta),
- **SMTP** (angl. *Simple Mail Transfer Protocol*), **POP** (angl. *Post Office Protocol*), **IMAP** (angl. *Internet Message Access Protocol*): različni protokoli za prenos elektronske pošte,
- **FTP** (angl. *File Transfer Protocol*): protokol za prenos datotek,
- **DNS** (angl. *Domain Name Service*): imenska storitev za preslikovanje imen strežnikov v IP naslove,
- **BitTorrent, Skype**: različni protokoli, ki tečejo med končnimi sistemi (P2P, angl. *peer-to-peer*).

Kako deluje protokol aplikacijske plasti zdaj že vemo. Po oblikovanju svojega sporočila zahteva aplikacijski protokol storitev prenosa tega sporočila od ene plasti, ki je nižja – transportna plast. Če govorimo o internetni aplikaciji, vemo, da lahko aplikacija za prenos uporabi enega izmed transportnih protokolov TCP ali UDP. Aplikacijski podatki se torej enkapsulirajo v takoimenovani *segment* TCP ali *datagram* UDP.

S slike 1.7 lahko vidimo, da aplikacijska plast modela TCP/IP združuje funkcionalnosti kar treh plasti modela ISO/OSI: aplikacijske, predstavitevne (ta določa pomen oziroma interpretacijo podatkov, npr. v okviru kodnih tabel in kompresije) in sejne plasti (ta skrbi za vzdrževanje več vzporednih/zaporednih povezav). Ker

model TCP/IP predstavitevne in sejne plasti eksplisitno ne definira, so naštete storitve v praksi običajno implementirane kar v aplikacijah samih.

V določenem omrežnem gostitelju se lahko nahaja več aplikacij, ki lahko komunicirajo preko omrežnih povezav. Zato mora (poleg naslavljanja računalnikov v omrežju) obstajati tudi način naslavljanja aplikacij, ki so jim dohodni podatki namenjeni. V poglavju 6 bomo podrobneje spoznali podrobnosti o delovanju aplikacijske plasti in pojem vrat (angl. *port*), ki se uporablja za naslavljanje aplikacij v omrežni napravi.

2 Fizična plast

Prav vsak podatek, od vsebine te knjige do vaše najljubše melodije in fotografije sosedovega kosmatinca, lahko zapišemo v digitalni obliki, vsak podatek lahko tudi predstavimo z elektromagnetnimi signali – tako z analognimi, digitalnimi kot tudi z drugimi fizikalnimi veličinami. V tem poglavju bomo razjasnili osnovne pojme – kaj pomeni digitalno ali analogno, kaj je sploh signal in kako komunikacijska povezava prenaša signale. Danes smo uporabniki navajeni, da zahtevamo čim hitrejši prenos podatkov, zato bomo pojasnili tudi, kaj pomeni pasovna širina povezave, slabljenje signala in kako pride do napak pri prenosu, saj vse našteto vpliva na hitrost in učinkovitost prenosa podatkov.

Uspeh pri prenosu podatkov je močno odvisen od kvalitete signala in od lastnosti prenosnega medija. Zato je cilj tega poglavja bralcu omogočiti občutek za pomen in vlogo teh dveh dejavnikov pri prenosu podatkov.

Kljud temu pa ne smemo pozabiti, da je temeljni cilj vsakega komuniciranja omogočiti povezavo med dvema uporabnikoma, torej je v ozadju na obeh straneh navadno človek. Tudi, ko s strežnika, ki se nahaja nekje globoko v drobovju interneta, prenašamo zadnjo različico protivirusnih datotek, jih je tja postavil človek z namenom, da jih k sebi prenesejo drugi.

2.1 Naloge fizične plasti

Predmet fizične plasti v računalniških omrežjih je predvsem preučevanje fizikalnih, točneje električnih lastnosti prenosnih medijev. Prenosni medij je lahko vsaka “naprava”, ki omogoča razširjanje valovanja, torej je primerna, da jo uporabimo za prenos signala od oddajnika do sprejemnika. Naloge fizične plasti so:

- preslikava logičnih ničel in enic v ustrezne fizikalne veličine (tok, napetost, frekvenca, faza...),
- pretvorba električnih signalov v obliko, ki je primerna za prenos po konkretnem prenosnem mediju,
- prenos signala po prenosnem mediju,
- določitev standardnih vmesnikov (konektorji - torej oblika in velikost vtičnic in vtikačev).

V nadaljevanju tega poglavja bomo posamezne naloge fizične plasti podrobno pojasnili in osmislili njihovo vlogo in pomen.

2.2 Osnovni pojmi na fizični plasti

Denimo, da želite priti k profesorju na govorilne ure in prositi, da vam prestavi rok za oddajo seminarske naloge na naslednji teden, saj res niste imeli časa, da bi jo dokončali že do jutri. Za začetek boste pogledali profesorjev zadnji status na Facebooku in presodili, ali je dovolj dobre volje, da se splača poskusiti. Pravzaprav to pomeni, da želite s profesorjem komunicirati – prenesti k sebi neke podatke, ki so z njegove strani postavljeni na ogled širnemu svetu, torej tudi vam, in na podlagi katerih boste morda lahko sklepali, kakšno je njegovo trenutno razpoloženje.

Da se ta komunikacija sploh lahko zgodi, se mora najprej zgoditi vrsta drugih povezav: kot uporabnik se povežete s svojo tablico prek zaslona, občutljivega na dotik, nato se povežete (prijavite) na Facebook. Ta povezava že sama po sebi zahteva vzpostavitev povezave vaše tablice z brezžično dostopno točko v vaši dnevni sobi, povezavo vaše dostopne točke oziroma domačega usmerjevalnika z usmerjevalnikom vašega ponudnika dostopa do interneta, in še kakih deset ali dvajset povezav med usmerjevalniki po celi svetu, preden paketek z vašim geslom sploh doseže Facebookov računalniški oblak.

Ko govorimo o prenosih na fizični plasti, radi zanemarimo kompleksnost svetovnega omrežja in se osredotočamo samo na povezavo med dvema sosednjima napravama. Nič zato, le občasno se bomo opomnili, da ne gre le za prenos ničel in enic, ampak da so na koncu vseh povezav uporabniki, ki željno čakajo na primer na najnovejšo spremembo statusa.

Pa se vrnimo k Facebooku in profesorjevemu statusu. Denimo, da se ta glasi: "Sonček sije, očitno končno prihaja pomlad! ☺" Ta status vsebuje dva podatka: prvi se nanaša na vreme v Ljubljani, drugi pa na dejstvo, da 21. marca nastopi meteorološka pomlad. Vendar znate sami ta dva podatka ustrezno interpretirati in na podlagi svojih dosedanjih izkušenj z ljudmi lahko sklepate, da je profesor dobre volje in da se splača poskusiti s prestavljivo termina za seminarsko nalogu.

Pa bodimo zdaj malo bolj formalni in zapišimo nekaj definicij:

- **podatek** je zapis dejstva (Sonček sije. Temperatura je 21° C.),
- **informacija** je interpretacija podatka in nastane v glavi uporabnika,
- **signal** je zapis podatka (ali dela podatka) v obliki, ki je primerna za prenos po prenosnem mediju,
- **komunikacija** pa je proces prenašanja podatkov. Namen tega prenašanja je shranjevanje ali pa interpretacija podatkov.

Naloga komunikacijskega sistema je, da omogoča komunikacijo, torej prenašanje podatkov. Naloga informacijskega sistema pa je, da obdeluje podatke (omogoča vnos, obdelavo, shranjevanje in prikaz podatkov) in s tem omogoča interpretacijo podatkov.

Prenašanje podatkov se izvaja med dvema entitetama, ki ju na fizični plasti imenujemo oddajnik in sprejemnik, podatki pa se zapisani v obliki signalov prenašajo po prenosnem mediju. V računalniških komunikacijah so navadno ti signali elektromagnetno valovanje, seveda pa bi bili v drugačnih okoliščinah primerni tudi razni drugi signali, tudi zvočni ali svetlobni signali, kot na primer Morsejeva abeceda ali pa navtična signalizacija z zastavicami – primer zadnje vidimo na sliki 2.1.

Prenosni medij lahko fizično usmerja signal od oddajnika do sprejemnika, primer takega medija sta bakrena žica in optično vlakno, lahko pa je tudi brezžični – pri tem se signal prenaša na vse strani, ne da bi ga posebej usmerjali in ne samo do nameravanega sprejemnika.



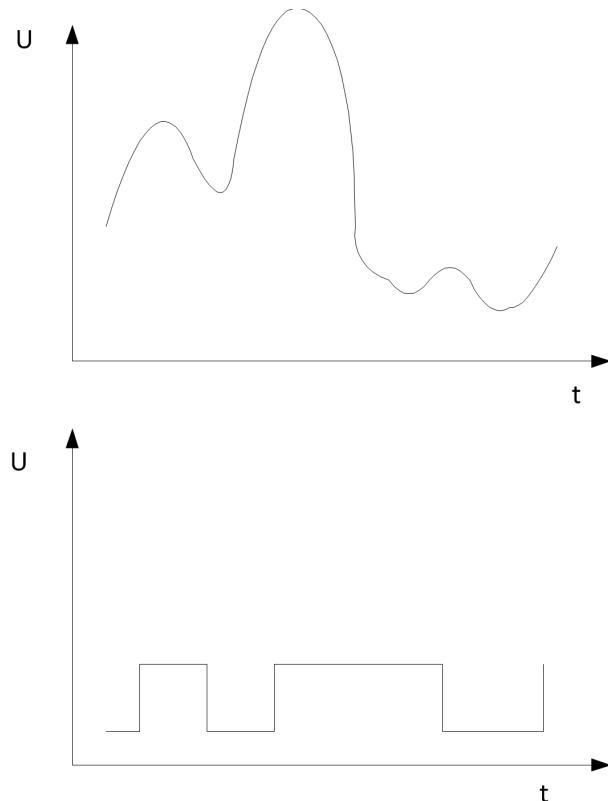
Slika 2.1 Nekatere zastavice - signali, ki ponazarjajo črke in številke v navtični signalizaciji

Prenos podatkov lahko poteka enosmerno (simplex – vedno v isto smer od istega oddajnika k istemu prejemniku – primer je oddajanje TV signala), izmenično dvosmerno (polovični duplex – v vsakem trenutku samo v eno smer, vendar se smer lahko spreminja – primer je walkie-talkie) ali pa popolnoma dvosmerno (duplex – naenkrat v obe smeri – primer je telefonska zveza). Danes je večina računalniških komunikacijskih povezav popolnoma dvosmernih.

2.3 O elektromagnetnih signalih

Elektromagnetne signale interpretiramo na različne načine. Če se signal s časom spreminja zvezno, pri čemer imajo za nas pomen vse njegove vrednosti od največje do najmanjše, takemu signalu rečemo **analogni** signal. Za digitalni signal pa znamo pomen pripisati le omejenemu diskretnemu naboru vrednosti, od katerih največkrat uporabljamo samo dve, ki ponazarjata ničlo in enico. Nivo signala v tem primeru nekaj časa ostane konstanten, nato pa hitro preskoči na

drug nivo, v idealnem primeru bi bila njegova oblika pravokotna. Primera obeh signalov prikazuje slika 2.2.



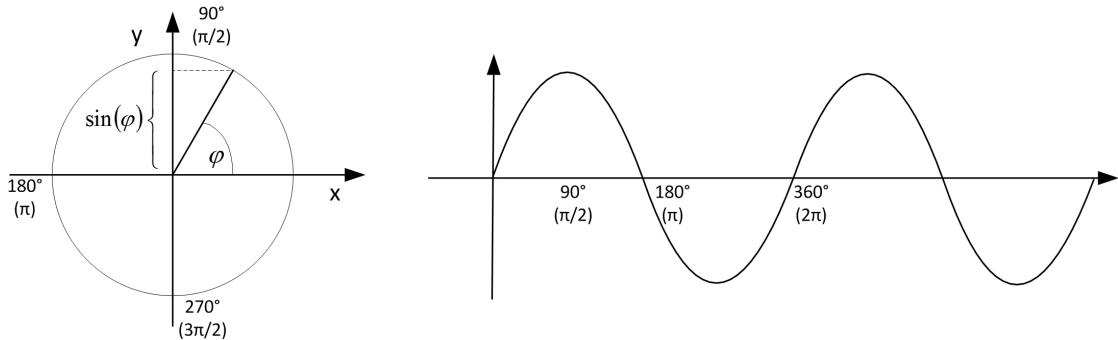
Slika 2.2 Če imajo vse vrednosti signala pomen, je signal analogen. Če lahko interpretiramo le diskretne vrednosti signala, na primer dve, je signal digitalen. Primer analognega (zgoraj) in idealnega digitalnega (spodaj) signala.

Pri periodičnem signalu se vzorec (vrednosti) s časom ponovi. Matematično lahko to zapišemo:

$$s(t + T) = s(t) \quad -\infty < t < \infty$$

Konstanta T predstavlja periodo signala, to je najmanjši časovni interval, po katerem se vrednosti signala ponovijo.

Osnovni in najpomembnejši periodični signal je sinusni signal. Spomnimo se na njegovo matematično definicijo: predstavljajmo si krog s polmerom 1 in središčem v koordinatnem izhodišču. Polmer kroga je enotska daljica z enim koncem v koordinatnem izhodišču, z drugim pa na krožnici. Glede na to, kje na krožnici je drugi konec, je lahko med to daljico in pozitivnim krakom abscise (vodoravne osi) kot od 0 do 360 stopinj. Funkcija $\sin(\varphi)$ predstavlja koordinato y ali drugače rečeno, projekcijo na ordinato (navpično os) točke na krožnici, v kateri polmer kroga s pozitivnim krakom abscise tvori kot φ , kot kaže slika 2.3.



Slika 2.3 Enotski krog in graf funkcije $\sin(x)$

Splošni sinusni signal poznamo, če poznamo njegovo amplitudo A , frekvenco f in fazo φ . Potem ga lahko opišemo s formulo

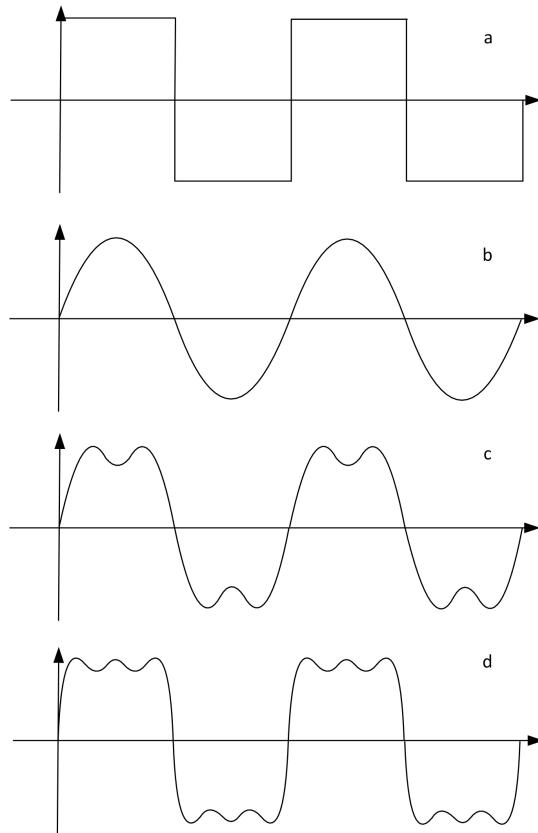
$$s(t) = A \sin(2\pi ft + \varphi).$$

Frekvanca predstavlja hitrost ponavljanja signala v ciklih na sekundo (Hz). Periода signala T pomeni čas, ki je potreben za eno ponovitev, zato velja $T = 1/f$. Po 360 stopinjah se vrednosti signala ponovijo. Signale, katerih frekvanca je večkratnik frekvence osnovnega signala, imenujemo višje harmonske komponente. Faza je merilo za položaj ali zamik periode po časovni osi. Merimo jo v stopinjah, enako kot kote. O sinusnem signalu brez zamika govorimo pri kotu 0 stopinj.

Resnični signali so sestavljeni iz mnogo frekvenc. S pomočjo Fourierove analize je možno pokazati, da je vsak (poljuben) signal sestavljen kot vsota sinusnih signalov z različnimi amplitudami, frekvencami in fazami. Množici frekvenc, ki tvorijo nek signal, rečemo spekter signala. Pasovna širina pa je razpon med največjo in najmanjšo frekvenco iz spektra. Čeprav imajo signali lahko tudi neskončno veliko pasovno širino, je večina energije signala vsebovana v relativno ozkem frekvenčnem pasu, ki mu rečemo efektivna pasovna širina.

V praksi se izkaže, da vsak prenosni medij lahko prenaša le omejen frekvenčni pas. Kaj to pomeni za oddani signal? Nekatere od frekvenc, ki ga sestavljajo, se preprosto izgubijo in dejansko se preneseni signal malenkostno razlikuje od oddanega.

Kot prikazuje slika 2.4, je preneseni signal bolj zvesta podoba izvornega signala, če je sestavljen iz več frekvenc. In obratno, več kot je frekvenc, ki jih prenosni medij ni spodoben prenesti, bolj je dejansko prejeti signal okrnjen in tako tudi različen od izvornega. Tudi idealni digitalni signal, kot na sliki 2.4 pod točko a, je možno zapisati kot vsoto signalov različnih frekvenc, le da je ta vsota pravzaprav neskončna vrsta. Če iz te vrste vzamemo le omejeno število členov, dobimo signale pod točkami b, c, ali d. Pri večji pasovni širini prenosnega medija tako lahko digitalni (pravokotni) signal prenesemo bolj zvesto ali pa si lahko



Slika 2.4 Digitalni signal (a) in njegovi približki: osnovna sinusna nosilna frekvanca (b), linearna kombinacija osnovne frekvence in njenega trikratnika (c) ter linearna kombinacija osnovne frekvence ter trikratnika (d)

privoščimo prenose enake kvalitete pri višjih frekvencah. Če je hitrost prenosa podatkov N b/s (bitov na sekundo), velja ocena, da bomo dobro reprezentacijo signala dobili pri pasovni širini $2N$ Hz. Če je hitrost pošiljanja višja kot N b/s, je potrebna večja pasovna širina od $2N$ Hz. To pravilo imenujemo Nyquistov teorem in ga bomo podrobneje pojasnili v nadaljevanju. Velja tudi obratno: Če je pasovna širina prenosnega medija večja, lahko podatke pošiljamo hitreje.

2.4 Digitalno in analogno

Besedi analogno in digitalno pogosto uporabljamo v zvezi s podatki, signali in s prenosom podatkov po povezavah.

Analogni podatki lahko zavzamejo na nekem intervalu vse možne vrednosti in vse imajo nek pomen. Primeri analognih podatkov so govor, glasba, svetlost, barva in podobno. Če so podatki diskretni, potem lahko zavzamejo le točno

določene vrednosti – na primer besedilo je lahko sestavljeno le iz črk od A do Ž in med črkama A in B nič vmesnega. Če diskretne podatke zapišemo z ničlami in enicami, govorimo o **digitalnih podatkih**. Besedilo na primer v digitalni obliki zapišemo tako, da vsako črko nadomestimo (kodiramo) z določenim zaporedjem ničel in enic.

Analogni signal se zvezno spreminja in vsaka njegova vrednost ima svoj pomen za podatke, ki jih prenašamo s takim signalom. Višja napetost na primer lahko pomeni glasnejši zvok, nižja pa tišji, če gre za prenos govora. **Digitalni signal** pa predstavlja zaporedje napetostnih pulzov. Tu določen čas trajajoča pozitivna napetost na primer lahko pomeni logično ničlo, negativna pa enico. Za prenos obeh vrst podatkov, analognih in digitalnih, lahko uporabimo tako analogni kot tudi digitalni signal, kot kaže tabela 2.1.

		vrsta uporabljenega signala	
		analogni signal	digitalni signal
Vrsta podatkov za prenos	analogni podatki (npr. govor)	Signal lahko zavzame enak frekvenčni spekter kot podatki ali pa podatke kodiramo s primernejšim delom frekvenčnega spektra. Primer: klasična analogna telefonska povezava.	Uporabimo vzorčenje in podatke zakodiramo v digitalno obliko. Primer: digitalna telefonska povezava (ISDN ali VoIP).
	digitalni podatki (npr. ASCII besedilo)	Iz digitalnih podatkov (npr. z modemom) poiščemo ustrezeno frekvenčno vrsto in generiramo analogni signal. Primer: prenos faksa po klasični analogni telefonski povezavi ali (zastareli) modemski dostop do interneta.	Logično ničlo in enico predstavljata dva različna napetostna nivoja ali tudi dve drugi ustrezeni fizikalni veličini. Primer: Prenos e-pošte po povezavi xDSL.

Tabela 2.1 Prenos digitalnih in analognih podatkov z digitalnim ali analognim signalom

Človeški glas (govor) zavzema frekvence približno od 300 Hz do 7000 Hz. Klasična analogna telefonska povezava lahko prenaša frekvence približno od 500 do 3600 Hz. Nižje od 500 Hz in višje od 3600 Hz se izgubijo in poslušalec zato sliši glas sogovornika rahlo popačen. Boljša hi-fi oprema zmore reproducirati tudi frekvence pod 100 Hz in nad 20.000 Hz. Človeško uho tega razpona ne sliši več (mladostnik morda še sliši visoke frekvence do 20.000 Hz, s starostjo pa ta sposobnost trajno upada in povprečen 40-letnik sliši le še do 10.000 Hz). V nadaljevanju si poglejmo, kako lahko človeški govor prenašamo v digitalni oblik. Najprej je potrebno govor, ki je po svoji naravi analogen signal, zapisati v digitalni oblik.

Za prenos digitalnih podatkov po analogni povezavi s pomočjo Fourierove analize signal zapišemo kot vsoto osnovne (nosilne) sinusne frekvence in njenih višjih harmonskih komponent. Čim več višjih komponent se lahko prenese, tem bolj kvaliteten bo prejeti signal. Nyquistov teorem pravi, da lahko analogni signal, ki ima najvišjo frekvenco N Hz, vzorčimo s frekvenco $2N$ (to pomeni, da zajamemo $2N$ vzorcev signala na sekundo) in bomo znali iz teh podatkov dovolj dobro obnoviti signal. Za vsak vzorec analognega signala tako opišemo amplitudo signala z digitalno vrednostjo, ki jo bomo potem lahko prenesli po digitalni povezavi.

Prenesimo povedano še v številke: če človeški govor vzorčimo z 8000 vzorci na sekundo, pomeni, da bomo dobro reproducirali frekvence do 4000 Hz. Dejiamo, da za vsak vzorec opišemo njegovo amplitudo z 8-bitno vrednostjo. Potem potrebujemo za prenos govora po digitalni povezavi 64.000 bitov na sekundo. V resnici se je tako vzorčenje uporabljalo že v prvih digitalnih telefonskih omrežjih pod imenom PCM (pulzno-kodna modulacija), danes pa podatke pred prenosom še stiskamo, saj s tem lahko dosežemo boljšo izkoriščenost prenosnega medija.

2.5 Neželeni električni pojavi pri prenosu

Na povezavi se med prenosom dogajajo neželeni električni pojavi:

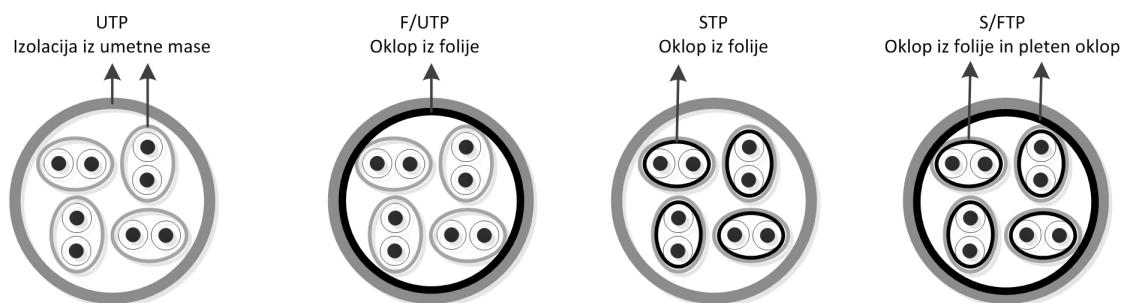
- **Slabljenje:** moč signala upada z razdaljo od oddajnika. V žicah in optičnih vlaknih je slabljenje eksponentna funkcija, ki jo lahko izrazimo kar v decibelih na enoto dolžine povezave. Na brezžičnih povezavah pa je to bolj kompleksna funkcija, odvisna od razdalje in tudi ozračja.
- **Popačenje:** v žicah in optičnih vlaknih je hitrost prenosa signala različna za različne frekvence signala. Pri določeni pasovni širini prenosnega medija je navadno najhitrejši prenos osrednje frekvence, proti višjim in nižjim frekvencam pa hitrost malenkost upade. Zaradi popačenja se nekatere frekvence prvega prenašanega bita prenašajo počasneje in se lahko prelijejo čez signal, ki prenaša že naslednji bit, in posledica tega je, da pride do interference med simboli.
- **Šum:** med prenosom signala na signal vplivajo tudi drugi, neželeni signali, ki prihajajo iz okolice – te imenujemo šum. Termični šum nastane zaradi termalne vzburenosti elektronov in je odvisen od temperature, prisoten pa je v vseh elektronskih napravah. Presluh je vpliv signala, ki se prenaša po sosednjih povezavah v neposredni bližini. Praktično vsaka naprava oddaja svoj značilen elektromagneten šum in vpliv vseh šumov iz okolja lahko občasno tako močno spremeni signal, da logična vrednost preskoči iz ničle v enico ali obratno.

2.6 Prenosni mediji

Z besedo prenosni mediji navadno razumemo vse vrste kablov, s katerimi lahko gradimo komunikacijska omrežja. Ne smemo pozabiti tudi na brezžične povezave, kjer prenos poteka po zraku ali v vesolju tudi v brezračnem prostoru. Pravzaprav ni razloga, da brezžični prenos ne bi potekal tudi po vodi ali po vaši najljubši piči, na primer pivu. Lahko rečemo, da je prenosni medij karkoli, po čemer se prenaša signal od oddajnika do sprejemnika. Lahko je snov, prostor, plin, tekočina, vakuum. V računalniških komunikacijah je najpomembnejša lastnost povezave za uporabnika zagotovo njena hitrost oziroma pravilneje kapaciteta: največje število prenesenih bitov v časovni enoti (b/s ali bps, kb/s, Mb/s, Gb/s).

Zvita parica. Zvita parica je najcenejši in obenem najširše uporabljen prenosni medij. Kot pove že ime, je sestavljena iz para - dveh izoliranih bakrenih žic premera 1 mm, ki sta naviti druga okrog druge s predpisanim številom ovojev na meter dolžine. To ovijanje prepreči ali zmanjša nekatere neželene električne pojave, zlasti presluh. Različice zvitih paric:

- UTP (unshielded twisted pair) je neoklopljena zvita parica, kot jo prikazuje slika 2.5 na levi strani. V omrežjih je kabel najpogosteje sestavljen iz štirih zvitih paric, ki so združene v skupnem izolativnem ovoju, v telefoniji pa se uporabljajo tudi kabli z 2, 4, 6, 25 in celo 100 pari žic.
- Oklopljena zvita parica je zelo podobna neoklopljeni, le da je na zunani strani dodatno zaščitena pred zunanjimi vplivi s kovinskim ovojem in / ali tanko ozemljitveno žico (screened). Različice so STP (shielded twisted pair), FTP (foiled twisted pair – ovoj iz tanke folije), S/UTP (screened UTP), S/STP ali S/FTP (screened shielded / foiled twisted pair), podrobnejše so prikazane na sliki 2.5.



Slika 2.5 Neoklopljena zvita parica UTP ima štiri zvite parice v skupnem ovoju, v sredini je lahko še distančnik. Oklopljena zvita parica F/UTP ima ovoj iz folije okrog vseh štirih parov. Pri oklopljeni zviti parici STP ima vsak par svoj ovoj, v S/FTP pa je ovoj iz prepletene žice okrog vseh štirih parov, od katerih je vsak ovit še v folijo.

Glede na frekvenčno območje ločimo več kategorij kablov in jih izbiramo glede na to, za kako hitro povezavo jih nameravamo uporabiti. Za omrežja Ethernet hitrosti 100 Mbit/s in 1Gbit/s lahko uporabimo kategoriji 5 ali 5e, obe zmoreta 100 MHz, čeprav za gigabitna omrežja raje posegamo po kategoriji 6, ki ima frekvenčno območje 250MHz. Za omrežja 10 Gbit/s pa je obvezna uporaba kategorije 6a (500 MHz). Za še hitrejša omrežja so že v pripravi standardi za kategoriji 7 (600 MHz) in 8 (1200 MHz). Kabel kategorije 5 je tipa UTP, 6 in 6a sta bodisi tipa UTP bodisi FTP, kategorija 7 pa je tipa S/FTP. Fizična plast določa tudi standardne konektorje za posamezno omrežno tehnologijo. Za kable UTP v omrežjih Ethernet se tako največ uporablja standard RJ45.

Dobre lastnosti zvite parice so predvsem njena tankost in prilagodljivost, saj je kable preprosto polagati, so upogljivi in jih lahko napeljemo okrog vseh vogalov, v komunikacijske kanale jih lahko nagnetemo kar veliko število, obenem pa je to najcenejši prenosni medij na enoto dolžine. Seveda ima tudi svoje slabosti. Če s kablom grobo ravnamo, se lahko poškoduje in nastajajo motnje, zelo je občutljiv tudi na zunanje motnje in elektromagnetne šume, relativno preprosto je tudi prisluškovanje.

Poleg različic UTP poznamo za bolj specializirana omrežja številne druge standarde, ki temeljijo na zvitih paricah. Kabel za hitro omrežje InfiniBand 4x (40 Gbit/s) na primer sestavlja 8 paric, vsaka v svojem ovoju in z več različnimi skupnimi ovoji, tudi pripadajoči konektor je popolnoma unikaten. Primer prikazuje slika 2.6.



Slika 2.6 Stikalo in konektorji za omrežje Infiniband

Koaksialni kabel. Koaksialni kabel je sestavljen iz dveh vodnikov: v sredini je prvi bakreni vodnik, ki je ovit v izolacijsko zaščito, okoli izolacije pa je prepletен

zunanji vodnik, ki obenem deluje tudi kot zaščitni oklop. Zunanji vodnik je ovit še z zunanjim plastično izolacijo, kot prikazuje slika 2.7. Tako nudi dobro zaščito pred zunanjimi elektro-magnetnimi vplivi in je relativno neobčutljiv za motnje. Koaksialni kabel oddaja zanemarljivo malo sevanja, zato je tudi prisluškovanje oteženo. Prenese pasovno širino do 2 GHz, torej ga lahko uporabljamo za hitre povezave. Žal pa je njegova cena visoka in tudi polaganje je zahtevno, saj je kabel relativno debel, tog in težak, ne prilagaja se dobro vogalom in zahteva več prostora kot UTP.



Slika 2.7 Zgradba koaksialnega kabla

Optika. Optični kabli so prenosni mediji, ki vsebujejo eno ali več optičnih vlaken. Jedro in ovoj sestavljata posebni vlakni, njun lomni količnik pa je izbran tako, da pride na meji med njima do popolnega odboja. Vse skupaj ščiti še dodaten plastični ovoj ali več plasti ovojev, odvisno od namembnosti kabla. Za težje pogoje uporabe je kabel položen še v zaščitno cev, lahko ga varuje zaščita iz kevlarja ali pa za vodo neprepustni plasč.

Na optični povezavi je slabljenje signala najmanjše, signal lahko prepotuje tudi 100 km brez ponavljalkov. Vendar pa so kabli občutljivi, mehanske poškodbe so pogoste in spajanje je zahtevno, saj je potrebno vlakno variti in na zvaru se lahko zgodi, da optične lastnosti, zahtevane za prenos svetlobe, niso več optimalne. Problematično je tudi upogibanje kablov, saj preveč upognjeno vlakno lahko poči.

Izvor svetlobe so LED diode ali polprevodniški laserji. Konektorji za optiko so različni, na vseh pa prihaja do izgub (10% do 20%). Po istem vlaknu lahko istočasno pošiljamo signale različnih valovnih dolžin oziroma barv svetlobe (temu pravimo, da signale multipleksiramo). Čeprav so tu določene omejitve glede valovnih dolžin in števila signalov, ki jih prenašamo istočasno, lahko tako dosežemo več prenosa v istem času. Na optiki lahko dosegamo hitrosti do 1 Tbit/s. V nasprotju s splošnim mnenjem, da optični povezavi ne moremo "prisluškovati", pa to ni nemogoče. Čeprav svetloba v okolico kabla ne seva

★ **Primer 2.1:** Golob pismonoša nosi okrog vratu obešeno pomnilno kartico SD kapacitete 1 GB. Če z njo v 20 minutah preleti predpisano pot, je njegov ekvivalent komunikacijska povezava kapacitete približno 7 Mbit/s. (Gigabajt je 8 gigabitov, gigabit pa je 1024 megabitov. Golobi opravijo pot v 20 minutah, se pravi v 1200 sekundah. Torej, koliko bitov na sekundo?)

tako močno kot električni signal, zaradi visokih frekvenc je namreč slabljenje močnejše, lahko napadalec odstrani na optični povezavi zaščitni plastični ovoj ter vlakno močno upogne. Na upognjenem mestu odboji znotraj vlakna niso več idealni in del svetlobe uhaja ven, kjer jo lahko napadalec prestreže, ojača in tako ulovi podatke, ki se prenašajo.

Brezžične povezave. Pri brezžičnih povezavah se elektromagnetni signal (radijski) prenaša po zraku ali brezračnem prostoru. Dobra lastnost takšnega prenosnega medija je, da je povsod že prisoten in ne zahteva nobene investicije, slabost pa so številni šumi in elektromagnetna onesnaženost, kar povzroča veliko napak pri prenosih. Nemogoče je tudi fizično varovanje prenosnih poti, saj se signal širi na vse strani. Glede na frekvenco signala brezžične povezave ločimo na radijske (Wi-Fi omrežja, Bluetooth, GSM), mikrovalovne (usmerjene povezave), infrardeče (zahtevajo vidno razdaljo in so namenjene le za kratke razdalje) in satelitske povezave, namenjene velikim razdaljam. Pri satelitskih povezavah je na satelit nameščena posebna naprava, imenovana transponder (ponavljalnik), ki na določenih frekvencah sprejema signal z zemlje, ga ojača in pošlje nazaj. Uporabljamo tri vrste satelitov: najvišje so geostacionarni sateliti na višini 36.000 km, na srednji višini so sateliti MEO (medium Earth orbit) na višini 18.000 km – na primer za GPS sistem, ter najnižji sateliti (LEO – low Earth orbit) na višini 750 km – takšni se uporabljajo na primer za sistem satelitske telefonije Iridium. Skupna značilnost vseh satelitskih povezav je visoka zakasnitev (latenca) približno 2 s pri prenosu podatkov.

Fizični prenos pomnilnih medijev. Podatke lahko prenašamo tudi na manj tehnološko zahteven način: shranimo jih na poljuben prenosni medij (USB ključek, CD, DVD, prenosni disk, magnetni trak, ...), sedemo na primer na kolo in jih sami prenesemo na drugi konec Ljubljane. Če vzamemo malo večji, a še vedno povprečen prenosni disk, lahko dosežemo prav impresivne hitrosti prenosa!

2.7 Kodiranje

V razdelku 2.4 smo pojasnili, da so podatki po svoji naravi lahko zvezni (analogni) ali nezvezni (diskretni) in da oboje lahko predstavimo in zapišemo v digitalni ali analogni obliki. Nekaj primerov smo tudi že predstavili v tabeli 2.1. Tako analogue kot tudi digitalne podatke je potrebno pred prenosom zakodirati v obliko analognih ali digitalnih signalov. Srečamo lahko štiri kombinacije:

- Digitalni podatki in digitalni signali: najpreprostejši način je, da logično ničlo in enico kodiramo z dverma različnima napetostnima nivojema. Primer: prenos računalniških podatkov po krajevnem omrežju Ethernet.
- Digitalni podatki in analogni signali: logično ničlo in enico zakodiramo tako, da nosilnemu analognemu sinusnemu signalu spremenjamo bodisi amplitudo, frekvenco ali pa fazo tako, da ena vrednost predstavlja ničlo druga pa enico. Primer: prenos računalniških podatkov po klasični analogni telefonski povezavi s pomočjo modema.
- Analogni podatki in digitalni signali: analogne podatke, na primer zvok, je potrebno pred prenosom digitalizirati, kar pomeni, da zvok v primernih časovnih intervalih vzorčimo (jemyemo vzorce signala, kot smo opisali v razdelku 2.4) in za opis posameznega vzorca uporabimo binarno število ustrezne dolžine. Primer: prenos telefonskega pogovora prek interneta.
- Analogni podatki in analogni signali: analogne podatke, na primer zvočni signal, lahko neposredno preslikamo v električni signal, kadar je to mogoče glede na naravo povezave. Še pogosteje pa podatke preslikamo na del frekvenčnega spektra, ki je najprimernejši za prenos. Primer: prenos telefonskega pogovora po klasični analogni telefonski povezavi.

V preostanku tega razdelka bomo nekatere zgoraj omenjene načine kodiranja pojasnili bolj podrobno.

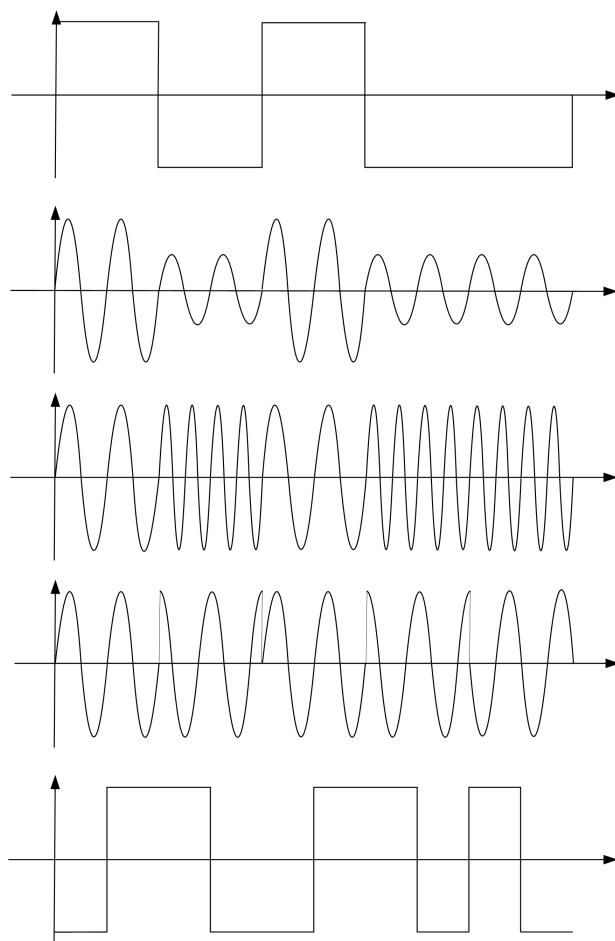
Modulacija. Modulacija predstavlja kodiranje podatkov s pomočjo analognega nosilnega signala. Zaradi številnih lepih lastnosti se kot nosilni signal navadno uporablja sinusni signal.

Pri **amplitudni modulaciji** ima nosilni sinusni signal ves čas enako frekvenco, podatke pa zakodiramo tako, da ena vrednost amplitude predstavlja logično enico, druga pa ničlo. Primer: višja amplituda je predstavlja enico, nižja pa ničlo. Če bi se tak signal prenašal po klasični analogni telefonski povezavi in bi prenos lahko poslušali, bi enico slišali kot glasnejše piskanje, ničlo pa kot tišje piskanje.

Pri **frekvenčni modulaciji** ima nosilni sinusni signal ves čas enako amplitudo, podatke pa zakodiramo tako, da ena frekvence pomeni logično ničlo, druga pa logično enico. Primer: frekvence N Hz pomeni ničlo, $2N$ Hz pa enico. Če bi prisluskovali takemu signalu in bi bila frekvence v slišnem območju, bi slišali

enico kot visok pisk, ničlo pa kot nižji pisk. Če bi bili to svetlobni signali v vidnem območju, bi videli različne barve.

Pri **fazni modulaciji** signalu sprememljamo fazo. Fazni kot signala nam pove, kako daleč od začetka periode smo v danem trenutku – faza 180° na primer pomeni, da smo ta hip na polovici periode, 90° pa da smo šele na prvi četrtini (glej sliko 2.3). Logično ničlo in enico tako lahko zakodiramo na primer s spremembama faznega kota nosilnega signala za 90 in 270 stopinj. Primere različnih vrst kodiranja prikazuje slika 2.8.



Slika 2.8 Kodiranje digitalnih vrednosti z analognimi signali in primerjava z digitalnim kodiranjem. Logične vrednosti 10100: zgoraj digitalni signal, ki uporablja kodiranje z dvema napetostnima nivojema, pod njim amplitudna, frekvenčna in fazna modulacija in na koncu Manchestersko kodiranje.

Spremembe faznega kota je možno na sprejemniku ob povprečni izgubi in motnjah signala sprejemati tudi z večjo razločljivostjo. Namesto da bi izbrali samo dve spremembi faze signala in z njima kodirali ničlo in enico, lahko izberemo

štiri različne spremembe in z njimi kodiramo štiri različne binarne vrednosti, torej vsaka sprememba faznega kota nosi informacijo dveh bitov, kot prikazuje tabela 2.2.

Sprememba faznega kota	Kodirana vrednost
0	00
90	01
180	10
270	11

Tabela 2.2 Primer kodiranja dvobitnih vrednosti s spremenjanjem faze po 90°

Lepa lastnost takšnega kodiranja je, da se po enaki povezavi z enako nosilno frekvenco lahko prenese dvakrat večja količina podatkov kot če bi uporabili amplitudno ali frekvenčno modulacijo z dvema nivojema.

Idejo o boljši izrabi razpoložljive pasovne širine pa lahko razvijamo še naprej. Poleg dveh različnih faznih kotov lahko še dodatno uporabimo amplitudno modulacijo in na ta način v eno spremembo signala zakodiramo še več podatkov. Kombinacijo imenujemo **kvadratna modulacija** (angl. *quadrature amplitude modulation – QAM*). Če uporabimo dva nivoja amplitudo in 4 spremembe faznega kota, lahko na primer z eno spremembo signala prenašamo 3 bite, kot kaže tabela 2.3.

Sprememba faznega kota	Amplituda	Kodirana vrednost	Diagram
0	nizka	000	
90	nizka	001	
180	nizka	010	
270	nizka	011	
0	visoka	100	
90	visoka	101	
180	visoka	110	
270	visoka	111	

Tabela 2.3 Primer kodiranja trobitnih vrednosti s spremenjanjem faze po 90° in z dvema nivojema amplitudo (kvadratna modulacija). Na diagramu vidimo iste podatke predstavljene še v grafični obliki – večja oddaljenost od središča pomeni višjo amplitudo.

Kodiranje z digitalnimi signali. Da lahko sprejemnik pravilno interpretira prejeti signal, mora vedeti, koliko časa traja predstavitev posamezne logične vrednosti - znati mora ugotoviti, kdaj se začne in konča posamezen bit. Nato pa mora ugotoviti, kakšna je vrednost signala v tem časovnem intervalu: na primer

visoka za logično ničlo in nizka za logično enico. To lahko doseže z jemanjem vzorcev signala: nekje na sredini intervala prisluhne signalu in zaznano vrednost nato primerja z vrednostjo praga, ki pomeni mejo med nivojem ničle in enice. Zaradi motenj in šumov lahko pride tukaj tudi do napak.

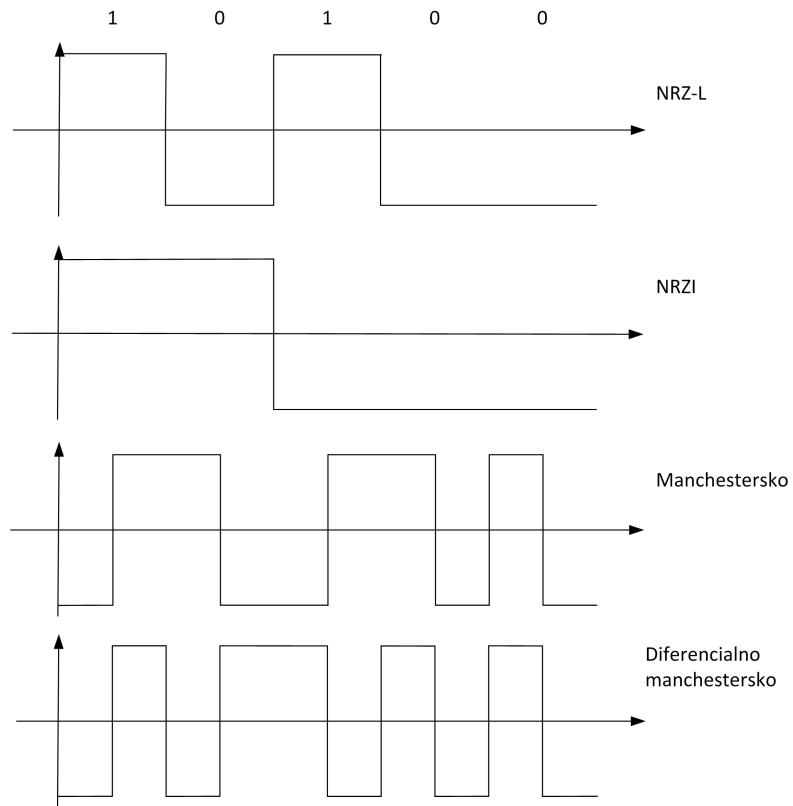
Pri najpreprostejšem kodiranju, kjer ničlo predstavlja visok nivo in enico nizek nivo, dolga zaporedja ničel ali enic pomenijo dolg čas trajanja nespremenjene vrednosti signala. V takem primeru se lahko zgodi, da se izgubi sinhronizacija med oddajnikom in sprejemnikom, kar v praksi pomeni, da sprejemnik ne ve več točno, kje v času je začetek in kje konec naslednje logične vrednosti, zakodirane v signal.

Kot alternativa, ki prepreči težave s sinhronizacijo, so se uveljavila kodiranja, kjer je v vsakem intervalu obvezen vsaj en prehod signala z visokega na nizek nivo ali obratno. Spremembu je namreč bolje zaznavna, torej naj bo v vsakem intervalu vsaj ena sprememba. Manchestersko kodiranje, ki se uporablja v omrežju Ethernet, ima tako v vsakem intervalu en prehod: logično ničlo predstavlja prehod iz nizkega na visoki nivo, logično enico pa prehod z visokega na nizki nivo. Če je torej signal v prejšnjem intervalu končal na nasprotnem nivoju, kot bi bil potreben za izvedbo prehoda, ki označuje naslednji bit, se zgodi še en prehod v začetku intervala.

Nekaj značilnih digitalnih kodiranj in primerjavo le-teh prikazujeta sliki 2.9 in 2.8. Na sliki 2.9 vidimo značilna digitalna kodiranja, ki jih opisujemo spodaj, na sliki 2.8 pa vidimo analogna kodiranja in je le za primerjavo dodano še manchestersko kodiranje.

- **NRZ-L** (Non Return to Zero-Level): brez prehoda na ničelni nivo (znotraj intervala posameznega bita).
 - 0: visok nivo
 - 1: nizek nivo
- **NRZI** (Non Return to Zero inverted): brez prehoda na ničelni nivo, z obračanjem
 - 0: na začetku intervala ni spremembe signala
 - 1: na začetku intervala se signal spremeni (če je nivo trenutno visok, postane nizek in obratno)
- **Manchestersko**: vsak interval vsebuje prehod
 - 0: na sredini intervala je prehod z visokega na nizek nivo
 - 1: na sredini intervala je prehod z nizkega na visok nivo
- **Diferencialno manchestersko**: na sredi intervala je vedno prehod.
 - 0: prehod je tudi na začetku intervala

- 1: na začetku intervala ni prehoda



Slika 2.9 Primerjava različnih digitalnih kodiranj (kodiranje digitalnih podatkov z digitalnimi signali)

2.8 Sklep

V tem poglavju smo pobliže spoznali osnove digitalnih komunikacij, ki poganjajo sodobni Internet. Čeprav smo temeljne teorije, koncepte in tehnike pojasnili dokaj intuitivno in se v matematično ozadje nismo spuščali pregloboko, smo nanizali dovolj podatkov za osnovno razumevanje fizikalnih zakonov in omejitev, ki krojijo okvire sodobnih komunikacij.

Ugotovili smo, da se prenos vseh vrst podatkov lahko izvaja s pomočjo elektromagnetskih signalov, opisali smo analogne in digitalne načine prenosa. Nato smo izpeljali, da poljuben signal sestavlja vrsta signalov različnih frekvenc in kot eno temeljnih lastnosti signala opredelili njegovo pasovno širino, torej razpon frekvenc, ki ga sestavljajo. Med prenosom po kateremkoli prenosnem mediju se signal vsaj malo popači, nanj vplivajo tudi šumi iz okolja. Zato smo primerjali

38 Poglavlje 2 Fizična plast

tudi prednosti in slabosti različnih prenosnih medijev in različnih načinov prenosa podatkov.

3 Povezavna plast

V prejšnjem poglavju smo spoznali fizično plast, ki določa način kodiranja bitov v signal in ga prenaša po prenosnem mediju – napravi, namenjeni razširjanju valovanja (bodisi elektromagnetnega ali svetlobnega). Fizična plast in povezavna plast, ki jo bomo podrobneje spoznali v tem poglavju, predstavlja skupaj *prenosni sistem*, ki je običajno implementiran v *omrežnem vmesniku* (strojni opremi).

3.1 Storitve povezavne plasti

Spomnimo se, da je računalniško omrežje sestavljeno iz velikega števila končnih sistemov in naprav, ki tvorijo jedro omrežja. To pomeni, da mora podatkovni paket pri potovanju med dvema končnima sistemoma prečkati več vmesnih omrežnih povezav. Glavna naloga povezavne plasti komunikacijskega sistema je prenesti zaokroženo zaporedje podatkovnih paketov po posamezni povezavi *med dvema sosednjima vozliščema*, upoštevajoč tip medija na tej povezavi. Če je torej pot med dvema končnima sistemoma sestavljena iz n povezav, se bodo na povezavni plasti med vsakim parom od $n + 1$ zaporednih naprav na tej poti odvile naloge, vezane na prenos po vmesni povezavi med njimi. Podrobneje, naloge, ki jih izvaja povezavna plast, so:

1. **Okvirjanje datagramov:** podatki višje (omrežne) plasti se opremijo z glavo (angl. *header*) in repom (angl. *trailer*) ter se jih enkapsulira v zaokroženo enoto, imenovano *okvir*,
2. **Zaznavanje in odpravljanje napak:** podatkom se pripšejo dodatni biti, s katerimi lahko ugotavljamo, ali je pri prenosu okvirja prišlo do napake; v določenih primerih lahko dodatne bite uporabimo tudi za to, da napako najdemo in jo popravimo,
3. **Dostop do medija:** če lahko komunikacijski medij uporablja več vmesnikov hkrati¹, izvaja povezavna plast protokol za dostop do medija (angl. *media*

¹Pravimo, da je takšen medij *deljen*. Primer deljenega medija so brezžične povezave

★ **Primer 3.1:** Za lažjo predstavo lahko način komunikacije preko več omrežnih povezav primerjamo s potovanjem po Sloveniji, za katerega potrebujemo več različnih prevoznih sredstev. Denimo, da se želimo iz Loma pod Storžičem napotiti v Koper in da za našo pot potrebujemo tri prevozna sredstva, ki so: taxi za pot od Loma do Kranja, avtobus za pot iz Kranja do Ljubljane in vlak za pot od Ljubljane do Kopra. Situacijo lahko primerjamo s prenosom podatka: naš potnik predstavlja *podatek*, ki se mora prenesti med začetnim krajem (*začetnim vozliščem*) in končnim krajem (*končnim vozliščem*), del poti predstavlja *komunikacijsko povezavo*, način prevoza po posameznem segmentu poti pa je analogen *protokolu povezavne plasti*. V opisani zgodbi manjka še nekdo, ki zna organizirati pot od začetnega do končnega kraja, o tem pa bomo govorili kasneje pri obravnavanju omrežne plasti.

access control (MAC) protocol²) in ustrezeno naslavljanje udeležencev v komunikaciji,

4. **Zagotavljanje zanesljive dostave:** uporaba potrjevanja in ponovnega pošiljanja v primeru napake na povezavi,
5. **Kontrola pretoka:** usklajevanje hitrosti pošiljanja glede na procesorske sposobnosti prejemnika.

V nadaljevanju si podrobneje poglejmo pomembnejše od zgornjih nalog.

3.1.1 Okvirjanje datagramov

Sedaj že vemo, da skrbi fizična plast za prenos logičnega zaporedja enic in ničel, ustrezeno zapisanih v obliko električnega ali optičnega signala. Zato, da prejemnik ve, kje se posamezno zaporedje bitov začne in konča, jih je potrebno zaokrožiti v enote, ki jih imenujemo **okvirji** (angl. *frame*). Vsak okvir ima natanko določen začetek in konec prenesenega zaporedja bitov ter vsebuje *glavo* in *rep* s podatki, potrebnimi za uspešen prenos.

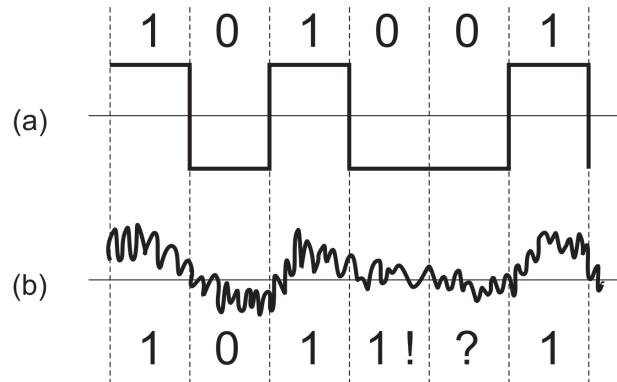
Ker komunikacija med dvema vozliščema v omrežju lahko poteka čez več omrežnih povezav, se lahko na teh povezavah uporablja različni *povezavni protokoli*. Na povezavni plasti poznamo veliko različnih protokolov, od katerih omenimo bolj znane: Ethernet, WLAN (802.11), Token Ring, PPP. Spomnimo se, da protokol opredeljuje zbirkovo pravil za komunikacijo, torej opredeljuje tudi način zapisa podatkov na vsaki od uporabljenih povezav.

3.1.2 Zaznavanje in popravljanje napak

Zaradi neželenih vplivov na komunikacijskem kanalu, kot so presluh, slabljenje in šum, se lahko signal pri prenosu pokvari. Če je okvara signala dovolj velika, lahko

²Razlikujmo kratico MAC za oznako protokola za dostop do medija od MAC naslova, ki ga bomo kasneje spoznali pri naslavljjanju vmesnikov na povezavni plasti.

prejemnik signal prebere drugače, kot ga je poslal pošiljatelj – torej zamenja enico in ničlo. Primer okvarjenega signala, ki povzroči napačno interpretacijo na strani prejemnika, prikazuje slika 3.1.



Slika 3.1 Motnje na kanalu lahko povzročijo, da prejemnik pri branju napačno prebere vrednosti poslanih bitov. (a) Idealno poslani signal s strani pošiljatelja. (b) Prejeti signal, ki je bil izpostavljen motnjam. Zaradi šuma lahko prejemnik zamenja vrednost četrtega (enica namesto ničle) in petega (nejasna vrednost) bita.

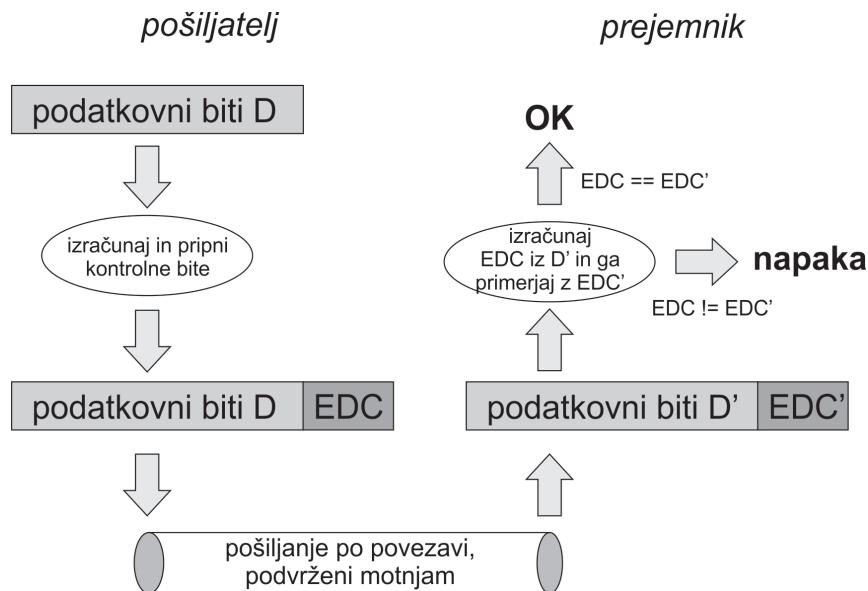
Komunikacijski sistem lahko na povezavni plasti skrbi za zaznavanje in popravljanje napak tako, da podatkom doda še dodatne *kontrolne bite* (angl. *error detection code, EDC*), namenjene preverjanju pravilnosti. Kontrolni biti niso sestavni del prvotnega zaporedja bitov – po prejemu se jih torej uporabi za preverjanje pravilnosti, nato pa zavrže. Zasnovno postopka zaznavanja napak prikazuje slika 3.2, ki prikazuje, kako lahko pride do napake pri prenosu podatkovnih ali dodatnih bitov, na povezavah, ki so podvržene napakam.

Kot primer algoritmov za zaznavanje in popravljanje napak omenimo paritetno, Hammingovo kodo in metodo CRC.

Paritetni biti

Dodajanje paritetnega (parnostnega) bita je ena od najbolj preprostih metod zaznavanja napak. Pri prenosu podatkov postopamo tako, da v zaporedju podatkovnih bitov preštejemo število enic k prvotnim podatkom zapišemo še en dodatni bit, ki predstavlja njihovo sodost ali lihost. Pred postopkom se moramo odločiti, kaj bo pomenila vrednost dodatnega pripetega bita, kjer imamo dve možnosti:

- če se odločimo, da vrednost 0 pomeni liho število enic in 1 pomeni sodo število enic, smo se odločili za *liho paritetno shemo*,
- če se odločimo, da vrednost 0 pomeni sodo število enic in 1 pomeni liho število enic, smo se odločili za *sodo paritetno shemo*.



Slika 3.2 Zasnova postopka zaznavanja napak pri prenosu z uporabo dodatnih bitov

★ **Primer 3.2:** Denimo, da želimo za zaporedje bitov

1	0	1	1	1	0	1	0
---	---	---	---	---	---	---	---

 izračunati paritetni bit. Iz niza vidimo, da ima niz liho število enic. Torej, če uporabljamo liho paritetno shemo, bo imel dodatni paritetni bit vrednost 0, če pa se odločimo za sodo paritetno shemo, bo paritetni bit enak 1.

S hitrim premislekom lahko ugotovimo, da z uporabo enega samega paritetnega bita lahko zaznamo samo neparno (liho) število napak. Namreč, če pri prenosu pride do dveh, štirih ali drugega sodega števila napak, se bo sodo/lihost števila enic ohranila in prejemnik napake ne bo zaznal.

Zgornjo težavo rešujemo z uporabo več paritetnih bitov, s katerimi merimo parnost v dveh dimenzijah. Ideja tega postopka je, da prvotno zaporedje podatkovnih bitov razdelimo v več enako dolgih vrstic in nato za vsako vrstico in stolpec izračunamo svoj paritetni bit. Tudi pri tem postopku lahko izberemo, ali uporabljamo sodo ali liho paritetno shemo, nad vsemi paritetnimi biti pa lahko izračunamo tudi meta-paritetni (ali kumulativni) bit. Iz primera na sliki 3.3 lahko vidimo, kako nam ta postopek omogoča zaznati in včasih celo popraviti enojne ali dvojne napake pri prenosu. Poleg zaznavanja napak v podatkovnih bitih lahko zaznamo tudi napake v prenesenih paritetnih bitih.

<i>poslano</i>	<i>prejeto</i>
1 0 1 1 1	1 (1) 1 1 0
1 0 1 0 0	1 0 1 0 0
0 0 0 1 0	0 1 0 1 0

Slika 3.3 Uporaba paritete v dveh dimenzijah. Pošiljatelj pošlje prejemniku niz 10111010, ki ga uredi v 2 vrstici in 4 stolpce ter za vsako vrstico in stolpec izračuna paritetni bit po sodi paritetni shemi. Pri prenosu se zapis drugega bita okvari, tako da prejemnik prejme niz 11111010, za katerega sam ponovno izračuna paritetne bite. Iz neujemanja drugega stolpičnega in prvega vrstičnega paritetnega bita (obarvana sta sivo) s prejetimi paritetnimi biti ugotovi, da je prišlo do napake v drugem bitu, ki jo lahko popravi.

Hammingova koda

Hammingovo kodo je predlagal Richard Hamming leta 1950 in omogoča zaznavanje napak pri največ dveh bitih ter popravljanje napak pri največ enem bitu. Od paritete v dveh dimenzijah je ta pristop boljši, ker uporablja manj dodatnih kontrolnih bitov. Hammingov postopek dodajanja kontrolnih bitov je sicer definiran za zaporedja poljubne dolžine bitov, za lažje razumevanje pa se omejimo na zaporedja 4 bitov:

- **Pošiljatelj:**

- prvotnemu 4-bitnemu nizu vrine tri kontrolne bite na 1., 2. in 4. mesto. Dobimo torej niz $p_1 p_2 n_3 p_4 n_5 n_6 n_7$, v katerem so p_i kontrolni biti, n_i pa biti prvotnega niza,
- določi p_1 tako, da je med biti $p_1 n_3 n_5 n_7$ sodo število enic,
- p_2 določi tako, da je med biti $p_2 n_3 n_6 n_7$ sodo število enic,
- p_4 določi tako, da je med biti $p_4 n_5 n_6 n_7$ sodo število enic.

- **Prejemnik:**

- prejme niz $p'_1 p'_2 n'_3 p'_4 n'_5 n'_6 n'_7$,
- določi $s_1 = 0$, če je med biti $p'_1 n'_3 n'_5 n'_7$ sodo število enic in $s_1 = 1$ sicer,
- $s_2 = 0$, če je med biti $p'_2 n'_3 n'_6 n'_7$ sodo število enic in $s_2 = 1$ sicer,
- $s_4 = 0$, če je med biti $p'_4 n'_5 n'_6 n'_7$ sodo število enic in $s_4 = 1$ sicer,
- Zaporedje bitov $s_4 s_2 s_1$, ki ga imenujemo *sindrom*, prejemniku v dvojškem sistemu pove, na katerem bitnem mestu je prišlo do morebitne okvare. Če je sindrom enak 000, je bil niz sprejet pravilno. V nasprotnem primeru lahko prejemnik popravi vrednost bita, ki ga podaja sindrom.

★ **Primer 3.3:** Denimo, da želi pošiljatelj prejemniku poslati zaporedje bitov 1011, ki ga želi opremiti s Hammingovo kodo. Pri postopku bo to zaporedje opremil s tremi dodatnimi biti p_1 , p_2 in p_4 , ki jih bo vrinil v zaporedje na mesta 1, 2 in 4, tako da dobi razširjen niz $\underline{0} \underline{1} \underline{0} 0 1 1$ (kontrolni biti so podčrtani). Kontrolni bit p_1 je pri tem določil iz zahteve, da mora biti med biti $p_1 \ n_3 \ n_5 \ n_7$ (torej med $p_1 \ 1 \ 0 \ 1$) sodo število enic; enako je določil tudi p_2 in p_4 .

Od tu naprej obravnavajmo dva scenarija. Denimo, da v prvem scenariju prejemnik pravilno sprejme poslani niz: $\underline{0} \underline{1} \underline{0} 0 1 1$. Na njegovi podlagi bo določil, da velja $s_1 = 0$, $s_2 = 0$ in $s_4 = 0$. Sindrom je torej enak 000, kar prejemniku pove, da pri prenosu niza ni prišlo do napake.

V drugem scenariju denimo, da prejemnik prejme niz, v katerem se okvari vrednost predzadnjega bita: $\underline{0} \underline{1} \underline{1} 0 0 1$. Na podlagi prejetega niza bo prejemnik določil $s_1 = 0$, $s_2 = 1$ in $s_4 = 1$. Sindrom ima torej dvojiško vrednost 110, ki je enaka vrednosti 6 v desetiškem sistemu. Ta vrednost prejemniku pove, da je prišlo do napake pri prenosu 6. bita. Ko prejemnik zamenja vrednost tega bita, popravi sprejeti niz na pravilni: $\underline{0} \underline{1} \underline{0} 0 1 1$.

Kontrolna vsota CRC

Kontrolna vsota CRC (angl. *Cyclic Redundancy Check*) je matematična metoda, ki je zasnovana na polinomih. Njena obravnavava presega okvire te knjige, zaznimo pa si lahko, da metoda uporablja n dodatnih kontrolnih bitov, s katerimi je možno zaznati in popraviti napake do $n + 1$ bitov. Pri obravnavi protokola Ethernet bomo videli, da le-ta pripenja 32-bitno kodo CRC repu vsakega okvirja in jo uporablja za zaznavanje (ne pa tudi za popravljanje) napak.

3.1.3 Dostop do medija

Kadar si več udeležencev v komunikaciji deli skupni medij, mora povezavna plast izvajati protokol za delitev tega medija med udeležence. Ravno tako mora poskrbeti za sistem naslavljanja udeležencev, da je nedvoumno jasno, komu je posamezen okvir namenjen.

Pri zasnovi protokola za dostop do skupinskega medija protokoli na povezavni plasti sledijo pravilom lepega komuniciranja, ki jih črpajo iz realnega sveta. Poglejmo si, kako se pravila bontona pri komuniciranju lahko navezujejo tudi na računalniške komunikacije:

- “*Daj vsakemu priložnost, da govori.*” → Vsakemu udeleženencu (komunikacijskemu sistemu) želimo omogočiti možnost uporabe skupinskega medija.
- “*Ne odgovarjaj, razen če te nekdo ne ogovori prvi.*” → Podatke pošiljaj le udeležencem, ki te za to prosijo in s tem nepotrebno ne zasedaj komunikacijskega kanala.

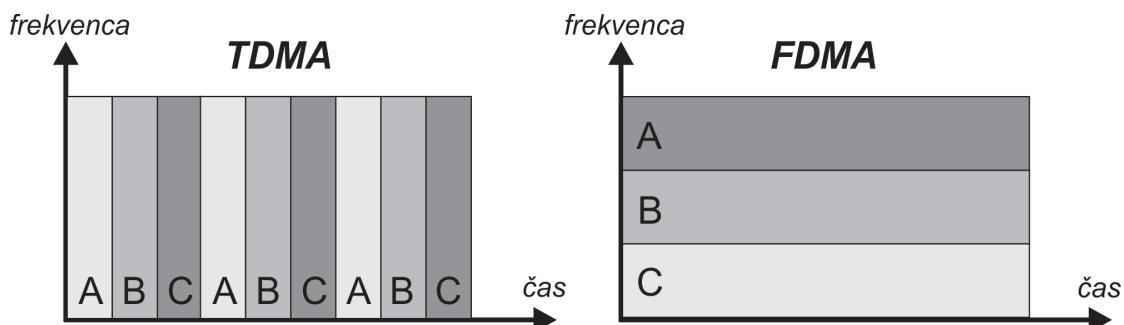
- *“Dvigni roko, če imaš vprašanje.”* → Počakaj, da prideš na vrsto za pošiljanje.
- *“Ne prekinjaj nekoga, ki govorí.”* → Počakaj na to, da ostali udeleženci v komunikaciji najprej zaključijo s svojimi prenosi in spregovori le takrat, ko je kanal prost.
- *“Ne spi, ko ti nekdo govorí.”* → Sprejmi podatek, ki si ga zahteval, da ga ne bo potrebno večkrat pošiljati.

Vidimo, da zgornje smernice “lepega vedenja” na skupnem komunikacijskem mediju veliko pozornost posvečajo temu, da lahko kanal istočasno uporablja največ en uporabnik. Če začne namreč naenkrat podatke pošiljati več udeležencev hkrati, pride namreč do pojava, ki ga bomo imenovali **trk** ali *kolizija* (angl. *collision*). V primeru trka se signali več udeležencev (ki so elektromagnetska valovanja) med sabo seštejejo ter se na ta način popačijo v prepletten signal, ki ga noben prejemnik ne more več pravilno interpretirati. Zato se želijo protokoli za dostop do skupinskega medija trkom izogniti ali pa čim bolj zmanjšati njihove negativne učinke nesmotrne rabe medija.

V nadaljevanju si poglejmo tri družine protokolov za dostop do skupinskega medija.

Protokoli za delitev kanala

Protokoli za delitev kanala delijo kanal na manjše dele, ki si jih lahko predstavljamo kot “podkanale”, od katerih je vsak namenjen enemu udeležencu v komunikaciji. Od najbolj znanih načinov za delitev kanala omenimo časovno delitev TDMA (angl. *Time Division Multiple Access*) in frekvenčno delitev FDMA (angl. *Frequency Division Multiple Access*). Oba načina delitve kanala prikazuje slika 3.4. Iz slike vidimo, da je pri časovni delitvi vsakemu pošiljatelju kanal izmenično na razpolago enako dolžino časa, pri frekvenčni delitvi pa, da ima vsak pošiljatelj na razpolago frekvenčni pas enake širine.



Slika 3.4 Časovna in frekvenčna delitev kanala med vozlišči A, B in C

Če želimo oceniti kakovost protokolov TDMA in FDMA, lahko rečemo, da sta oba učinkovita in pravična (vsakemu udeležencu omogoča enako izrabo kapacitete) pri visoki obremenitvi kanala, saj dosegata popolno izrabo kapacitete kanala pri popolnem izogibanju trkom. Po drugi strani pa vidimo, da če želi v komunikaciji sodelovati eno samo vozlišče, le-to ne more izrabiti polne kapacitete kanala, saj je ta nespremenljivo deljena med število uporabnikov.

Protokoli za naključni dostop

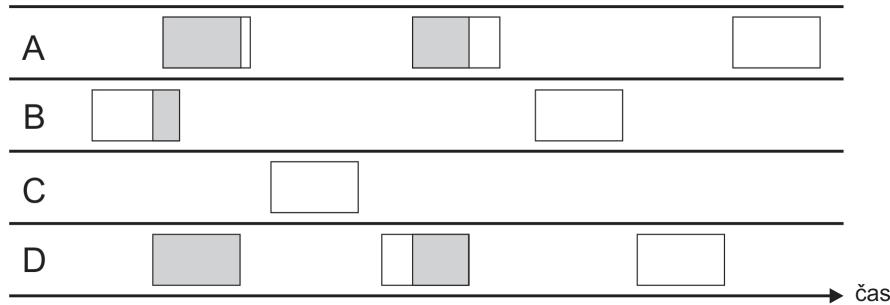
Protokoli za naključni dostop dovoljujejo, da lahko vsako vozlišče pri pošiljanju uporabi polno kapaciteto kanala. Protokoli določajo, kdaj lahko pošiljatelj začne pošiljati, kako zaznati, ali je prišlo do trka, in kako ob njem ukrepati.

ALOHA. Med prvimi protokoli za dostop do medija je bil protokol ALOHA, ki je nastal v zgodnjih 70. letih. Uporabljal se je na Havajih za povezovanje oddaljenih otoških računalnikov z glavnim računalnikom na otoku Honolulu. Protokol dovoljuje, da lahko uporabnik pošlje okvir, kadarkoli želi. Ker lahko posledično pride do trka dveh ali več okvirjev, se lahko okvirji pri prenosu okvarijo in potrebno jih je poslati ponovno. Pred ponovnim pošiljanjem mora vsak pošiljatelj, ki je povzročil trk, počakati naključno dolg časovni interval, preden okvir ponovno pošlje (če bi pošiljatelja morala počakati enako dolgo, bi okvirja ponovno trčila).

Primer uporabe protokola ALOHA med tremi udeleženci prikazuje slika 3.5. Iz slike vidimo, da je čas obravnavan zvezno, kar pomeni, da lahko vsak pošiljatelj povzroči trk tudi takrat, ko je večji delež nekega drugega okvirja poslan že skoraj do konca. Protokol narekuje, da naj pošiljatelj, ne glede na prisotnost trka, svoj okvir vedno pošlje do konca. Ugotavljanje, ali je prišlo do trka, je bilo izvedeno tako, da je centralni računalnik uspešno sprejete okvirje poslal nazaj računalnikom, ki so jih prej oddali. Če so pošiljatelji torej ponovno prejeli svoj okvir, so s tem dobili potrditev o uspešnem sprejemu njihovega okvirja.

Trki znižujejo učinkovitost prenosa, saj je potrebno oba okvirja poslati še enkrat. Okvir med prenosom nikoli ni varen: neka druga naprava lahko sproži prenos kadarkoli, takrat pa se na vodilu zasliši nov signal in trk je tu. Pri protokolu ALOHA je učinkovitost nizka, v primeru velike obremenjenosti pade na samo 18%. To pomeni, da je samo 18% časa prenosni medij uporabljen za uspešne prenose (brez trkov), vseh ostalih 82% časa pa se prenašajo okvirji, ki nato doživijo trk in jih je potrebno prenesti ponovno.

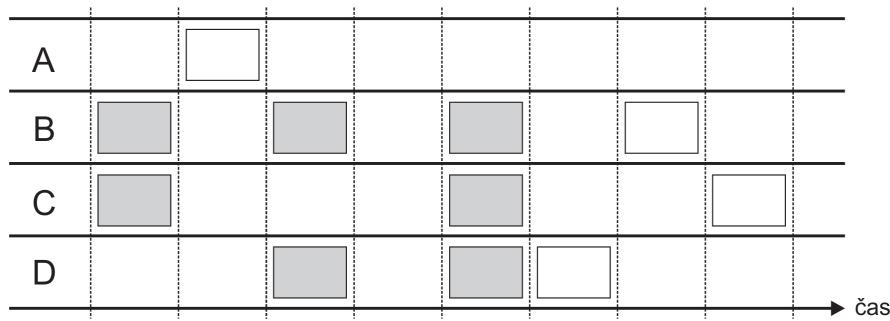
Razsekana ALOHA. Razsekana ALOHA (angl. *slotted ALOHA*) povečuje učinkovitost prenosa tako, da deli čas na diskretne, enako dolge časovne intervale, v katerih je možno poslati natanko 1 okvir. Protokol narekuje, da lahko vozlišča svoj okvir oddajo samo ob začetku teh intervalov, kar v povprečju prepolovi čas, ko je okvir na mediju ranljiv za trke s strani drugih okvirjev. V primeru, da pride



Slika 3.5 Protokol ALOHA. Štiri vozlišča želijo oddati okvirje na skupni medij. Ker prihaja do trkov (sivi deli okvirjev označujejo časovne intervale, ko je na mediju več signalov hkrati, torej obdobje trka), morajo vozlišča A, B in D okvir poslati ponovno, po preteku naključno dolgega časovnega intervala. Vozlišču C uspe oddati okvir v prvem, vozlišču B v drugem, vozliščema A in D pa v tretjem poskusu.

do trka, pošlje vozlišče svoj okvir v vsakem naslednjem časovnem intervalu z neko verjetnostjo p , s katero oponaša čakanje naključnega števila intervalov, vse dokler prenos končno ne uspe.

Primer izvajanja protokola razsekana ALOHA je prikazan na sliki 3.6. Prednost te različice protokola je, da lahko vsak udeleženec uporablja polno kapaciteto kanala. Slaba stran protokola je, da si lahko kanal v nekem časovnem intervalu udeleženci lahko pravično razdelijo le v najboljšem primeru – če ne pride do nobenega trka; v vseh preostalih primerih je učinkovitost pošiljanja zaradi trkov manjša, a vendarle večja kot pri prejšnji različici ALOHA in znaša 37%. Protokol ima še dodatno oviro: enostavnost njegove implementacije je vprašljiva, saj potrebuje zelo natančno časovno sinhronizacijo vseh udeležencev zaradi usklajevanja intervalov za pošiljanje okvirjev.



Slika 3.6 Razsekana ALOHA. Štiri vozlišča želijo oddati vsako svoj okvir na skupni medij. Ker prihaja do trkov (sivi okvirji), morajo vozlišča B, C in D okvir poslati ponovno.

Carrier Sense Multiple Access (CSMA). CSMA je sodoben in naprednejši protokol od dosedaj naštetih, ki se danes uporablja na številnih ožičenih in brezžičnih deljenih medijih. Besedi *multiple access* v nazivu protokola označuje, da gre za protokol za nadzorovanje dostopa do skupinskega medija; besedi *carrier sense* pa opisujeta, da gre za protokol, ki je sposoben zaznavanja, ali katero od drugih vozlišč že oddaja svoj okvir. Po domače zapisano, ta protokol sledi principu: *"Prisluhni, ali že kdo drug govor, preden spregovoriš tudi ti"*.

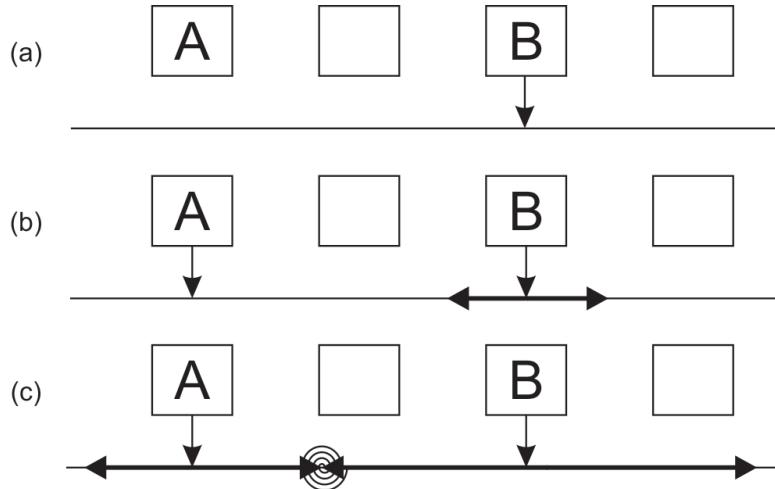
Vozlišče, ki čaka, da se medij sprosti, lahko uporablja različne različice protokola CSMA. Če vozlišče neprekinjeno posluša medij, vse dokler se ta ne sprosti, pravimo, da gre za **vztrajni CSMA** protokol (angl. *persistent CSMA*). Za razliko od tega **nevztrajni CSMA** protokol (angl. *non-persistent CSMA*) "manj požrešno" preverja prostost medija in mu večkrat prisluhne po preteklu naključnih časovnih intervalov. Ko je medij prost, vozlišče seveda takoj odda svoj podatkovni okvir. Poznamo tudi **p-vztrajni CSMA** (angl. *p-persistent CSMA*), kjer p označuje verjetnost, s katero vozlišče odda okvir v časovnem intervalu, ko zazna, da je medij prost. Za vztrajni CSMA lahko torej rečemo, da je enak 1-vztrajnemu CSMA ($p = 1$).

Kljub temu, da vozlišče pred oddajanjem prisluhne mediju, pa do trkov lahko vendarle pride. Kje tiči razlog za to? Omrežni vmesnik potem, ko zazna prost medij, potrebuje določen čas, da zgenerira ustrezni signal in ga pošlje na komunikacijsko povezavo. Če v tem času mediju prisluhne druga naprava, ga bo zaznala kot prostega in bo tudi ona lahko zgenerirala in oddala svoj signal, tako da bo med njima prišlo do trka. Vendar pa se trk lahko zgodi le v začetku oddajanja okvirja. Ko se okvir nekaj časa oddaja in medij torej že nekaj časa ni prost, ni več nevarnosti, da bi ga katera naprava lahko zaznala kot prostega in začela oddajati.

Čas potovanja signala od povezavne plasti pošiljatelja do povezavne plasti prejemnika je torej večji od nič, imenujemo pa ga *propagacijska zakasnitev* (angl. *propagation delay* ali *zakasnitev pri razširjanju signala*). Zaradi propagacijske zakasnitve lahko pride do pojava, ko lahko eno vozlišče zazna, da je medij prost, medtem ko je signal drugega vozlišča že prisoten na mediju, a ni še priproval do prvega vozlišča. Shemo takšnega scenarija prikazuje slika 3.7.

Carrier Sense Multiple Access/Collision Detection (CSMA/CD). CSMA/CD je izboljšava osnovnega CSMA, saj vozlišča pri slednjem, kljub trku, vedno pošljejo svoje okvirje do konca. Na ta način je komunikacijski kanal v času trka izrabljen neučinkovito, saj je popačene okvirje potrebno zavreči, kanal v času njihovega pošiljanja pa ni bil prost za druga, morebitno uspešna pošiljanja.

Vozlišča, ki uporabljajo protokol CSMA/CD ves čas med oddajanjem zaznavajo, ali je morda prišlo do trka - to lastnost imenujemo zaznavanje trkov (angl. *Collision Detection*). Če zaznajo trk, takoj prenehajo z oddajanjem tre-



Slika 3.7 Trk zaradi propagacijske zakasnitve. (a) Vozlišče B prisluhne mediju (*carrier sense*), da vidi, ali je prost; (b) Vozlišče B začne oddajati signal. Še preden signal prišteje do vozlišča A, tudi A prisluhne mediju in začne oddajati svoj okvir; (c) Na mediju pride do trka okvirjev vozlišč A in B.

nutnega okvirja, saj je že jasno, da prenos ne bo uspel. Po domače bi lahko rekli, da sledijo principu: *„Če vskočiš komu v besedo, čimprej nehaj govoriti.“*. Za uspešno zaznavanje trkov morajo vozlišča torej med pošiljanjem nenehno poslušati medij. Za nedvoumno ugotovitev, da na mediju prihaja do trka, potrebuje vozlišče določen čas (angl. *collision detection/abort time*).

Protokol Ethernet, ki je danes najbolj razširjen protokol na povezavni plasti v krajevnih (lokalnih) omrežjih, uporablja različico protokola CSMA/CD. Poleg osnovnih nalog tega protokola, Ethernet vozlišča ob zaznanem trku drugim vozliščem pošljejo *motilni signal* (angl. *jam signal*), s katerim okvarijo okvirje, ki so jih druga vozlišča že delno sprejela ter s tem omogočijo hitro in nedvoumno zaznavo trka tudi ostalim vozliščem na skupnem mediju.

Carrier Sense Multiple Access/Collision Avoidance (CSMA/CA). Protokol CSMA/CA je protokol za naključni dostop, ki se pogosto uporablja v brezžičnih omrežjih. V brezžičnih omrežjih je sprejem signala pogojen z oddaljenostjo, okoljem in motnjami. Vozlišča imajo lahko različna območja pokritosti s svojim signalom in slišijo jih lahko le naprave, ki se nahajajo znotraj tega območja. Tudi zaznavanje zasedenosti medija (angl. *Carrier Sense*) ni zanesljivo. Dve napravi sta lahko ena drugi zunaj dometa signala in obe mislita, da je medij prost, saj ne zaznavata prenosa druge druge. Tretja naprava pa je na sredi med njima, v območju dometa obeh njunih signalov, in na tem območju se dogaja trk. Tretja naprava torej zaradi trka ne more sprejemati podatkov od nobene od njiju.

Zaznavanje trkov (angl. *Collision Detection*) torej ni izvedljivo, saj zaradi slabljenja signala skozi prostor na mestu pošiljatelja težko sklepamo, kaj se dogaja v okolici; istočasno pa pošiljatelj tudi ne more pošiljati podatkov in zaznavati stanja na mediju³. Z drugimi besedami, lastnost brezžičnega medija zagotavlja pošiljatelju le, da zaznava druge pošiljatelje v svoji bližini, ne more pa zaznavati morebitnih trkov na mestu prejemnika.

Namesto zaznavanja trkov je naloga protokola CSMA/CA torej preventivno delovanje - da se jim poskuša izogniti. Za ta namen protokol uporablja dve protokolarni sporočili (okvirja posebne oblike), ki nadzorujeta rezervacijo medija: RTS (angl. *Request To Send*) in CTS (angl. *Clear To Send*).

Vozlišče, ki želi oddati okvir (pošiljatelj), mora pred oddajo dobiti dovoljenje od prejemnika. S sporočilom RTS ("Želim ti poslati okvir!") pošiljatelj zaprosi za dovoljenje za pošiljanje; v isto sporočilo zapiše tudi, kakšno količino podatkov želi poslati. Ko je prejemnik pripravljen na poslušanje, mu pošlje dovoljenje - sporočilo CTS ("Izvoli, pošiljaj!"). Vanj zapiše tudi, komu je dovoljenje namenjeno in koliko časa bo trajalo pošiljanje oziroma koliko podatkov se lahko pošlje. Sporočilo CTS slišijo tudi druga vozlišča, ki so v dometu prejemnika. Ker ta preostala vozlišča niso zahtevala komunikacije z vozliščem B, razumejo prejeto sporočilo CTS kot znamenje, naj obmolknejo oziroma ne oddajajo v času prenosa toliko podatkov, kolikor je bilo zapisano v sporočilu CTS, ki so ga prejeli. Vozlišče A lahko torej v tem času uspešno pošlje podatkovni okvir vozlišču B in se predvidoma izogne trkom z okvirji preostalih vozlišč.

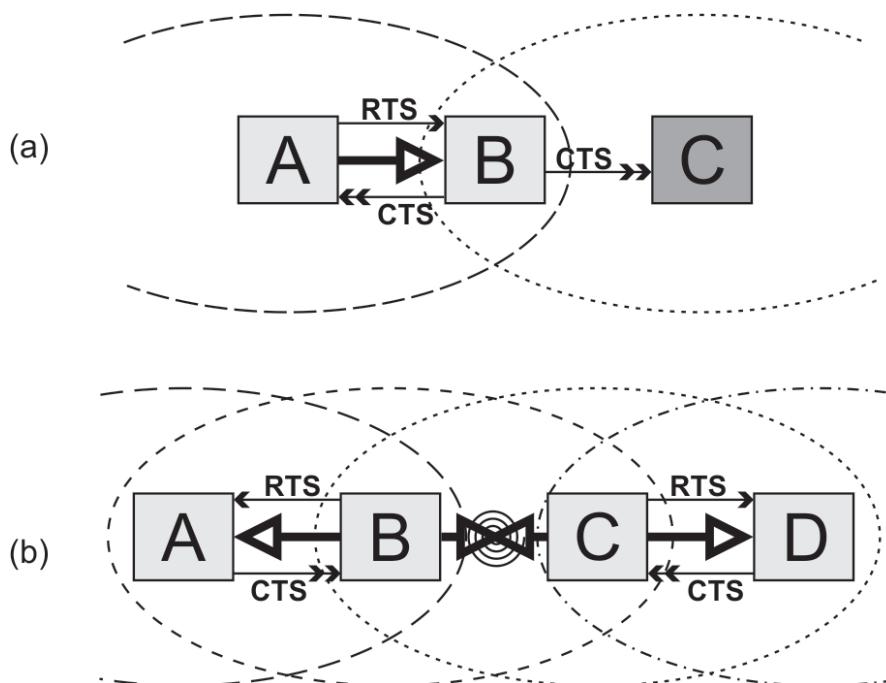
Čeprav mehanizem izogibanja trkom prepreči trke podatkovnih okvirjev, pa so trki v omrežju, ki uporablja CSMA/CA, še vedno možni. Kot zelo preprost primer podajmo situacijo, kjer si dve vozlišči istočasno vzajemno pošljeta signala RTS, ki se bosta posledično pri prenosu okvarila (ta okvara sicer ne pomeni nič dramatičnega, saj po preteku določenega časa vozlišče, ki ne prejme dovoljenja za pošiljanje CTS, preprosto ponovno odda zahtevo za pošiljanje RTS). Zato brezžični protokoli uporabljo sistem sprotnega potrjevanja prejetih okvirjev in ponovnega pošiljanja, saj tako zagotavljajo večjo zanesljivost dostave. Primer uporabe CSMA/CA in potrjevanja je tudi protokol IEEE 802.11 (WiFi).

Uporaba izogibanja trkom in potrjevanja sta opcionalni (predmet nastavitev). Če ju ne uporabljamo, popačeni paketi enostavno manjkajo in se z njihovo odsotnostjo in ponovnim pošiljanjem (če je to potrebno) ubadajo višje plasti, ponavadi transportna ali aplikacijska.

Prostorska razporeditev vozlišč igra veliko vlogo pri uspešnem preprečevanju trkov, zato si v nadaljevanju oglejmo dva vzorčna primera razporeditev, pri katerih uspešna komunikacija brez uporabe RTS in CTS ne bi bila možna (obe razporeditvi vozlišč sta prikazani na sliki 3.8). Težavo, ki jo prikazuje prva

³Zaradi teh omejitev je protokol v literaturi pogosto poimenovan tudi kot MACA - *Multiple Access with Collision Avoidance*

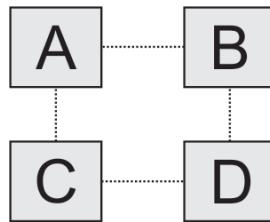
postavitev, imenujemo **skriti vozlišči**) (angl. *hidden terminals*), pri kateri vozlišči A in C nista v medsebojnem dosegu, pošiljati pa želita vozlišču B, ki se nahaja v dosegu obeh. A in C se med seboj ne slišita in ne zaznavata trka njunih signalov. Če istočasno oddajata, pa vendarle povzročita trk v vozlišču B. Slika 3.8(a) prikazuje, kako uporaba sporočila CTS prepreči istočasno pošiljanje s strani vozlišč A in C: Vozlišče A z RTS zaprosi B za dovoljenje za pošiljanje; nato B odda dovoljenje (CTS), ki je slišno tudi vozlišču C in ta iz njega razbere, da nekaj časa ne sme oddajati podatkov.



Slika 3.8 Dve tipični postavitev brezžičnih vozlišč (elipse ponazarjajo doseg vozlišča, ki je v centru elipse). (a) **Postavitev skritih vozlišč**. Signal CTS, govori vozlišče B, vzpostavi komunikacijo (svetlo siva) med A in B in prepreči pošiljanje (temno siva) s strani C ter s tem povzročitev trka pri B. (b) **Postavitev izpostavljenih vozlišč**. Signala CTS vozlišč A in D omogočita istočasno komunikacijo teh vozlišč z vozliščema B in C. Čeprav prihaja do trkov v prostoru med B in C, to ne vpliva na komunikacijo v izvajaju, saj kraj kolizije ni v dosegu robnih vozlišč.

Drugo tipično postavitev imenujemo postavitev **izpostavljenih vozlišč** (angl. *exposed terminals*). V tem scenariju so 4 vozlišča A, B, C in D razporejena tako, da ima vsak v dosegu samo najbližjega levega in najbližjega desnega soseda (skrajni vozlišči A in D pa imata v dosegu samo eno sosednje vozlišče). Ker sta vozlišči B in C v medsebojnem dosegu in ob istočasnem pošiljanju povzročata trke v medsebojnem prostoru, pravimo, da sta izpostavljeni eno drugemu. Pri tej postavitev se pojavlja težava, če želita vozlišči B in C istočasno komunicirati:

★ **Primer 3.4:** Za vajo analizirajmo sistem postavite brezžičnih omrežnih vozlišč, ki je prikazan na spodnji sliki (doseg med vozlišči je označen s črtkanimi povezavami namesto z elipsami). S slike vidimo, da sta v dosegu vsakega izmed štirih vozlišč dve preostali vozlišči. Zastavimo si in odgovorimo na dve vprašanji:



1. *Ali lahko medtem, ko vozlišče A pošilja vozlišču B, tudi vozlišče D pošilja vozlišču C?* Odgovor je *ne*. Namreč, če vozlišče A pošilje RTS vozlišču B, bosta CTS vozlišča B slišala vozlišči A in D. D zato v času, ko A pošilja, ne bo oddajal.
2. *Ali lahko medtem, ko vozlišče A pošilja vozlišču B, tudi vozlišče C pošilja vozlišču D?* Odgovor je *da*. CTS, s katerim odgovori vozlišče B vozlišču A, slišita samo vozlišči A in D, ne pa tudi vozlišče C. Vozlišče C lahko zato vozlišču D pošilje okvir, s katerim ne bo zmotil B-jevega sprejema, kljub temu, da sta si vozlišči A in C izpostavljeni.

B z vozliščem A, C pa z vozliščem D. Med njima se bo namreč pojavil trk, ki ga A in D sicer ne bosta zaznala in bosta sprejemala nepopačene podatke. Če ne bi uporabljali izogibanja kolizijam, bi eno od vozlišč pošiljalo, drugo pa bi zaradi trka med vozliščema B in C ugotovilo, da je medij zaseden, in bi čakalo, da se sprosti. S slike 3.8(b) vidimo, da uporaba sporočil RTS in CTS omogoči istočasno pošiljanje vozlišč B in C, saj oba okvirja CTS dosežeta le eno najbližje vozlišče. To pomeni, da tudi v primeru izpostavljenih vozlišč protokol CSMA/CA zagotavlja uspešen prenos, saj trk med vozliščema B in C v danem scenariju ni pomemben.

Protokoli za izmenični dostop

Protokoli za izmenični dostopa vozliščem izmenično dodeljujejo pravico do pošiljanja. Pri njih torej do trkov ne more priti, dodaten režijski čas pa se namesto za trke porablja za fazo dodeljevanja pravice pošiljanja. Na hitro omenimo dva pristopa k zasnovi te vrste protokolov.

Poizvedovanje (angl. *polling*). Poizvedovanje se izvaja s strani centralnega vozlišča, ki zaporedoma sprašuje vozlišča, priključena na medij, ali želijo poslati podatke. Če želi vozlišče poslati podatke, lahko to naredi samo takrat, kadar je

na vrsti. Povpraševanje s strani centralnega vozlišča doprinaša k manj učinkoviti izkoriščenosti medija, istočasno pa ima celoten sistem tudi enotno točko odpovedi – centralno vozlišče.

Podajanje žetona (angl. *token passing*). Podajanje žetona je posebna oblika poizvedovanja, ki ne uporablja centralnega vozlišča, temveč posebno sporočilo, imenovano **žeton**, ki vozlišču dodeljuje pravico do pošiljanja. Žeton kroži med vozlišči po v naprej določenem zaporedju. Vozlišče, ki ima žeton pri sebi, ima pravico oddati svoj okvir, nato pa mora žeton posredovati naslednjemu vozlišču. Če ne želi oddati okvirja, žeton takoj posreduje naprej.

Specifična oblika podajanja žetona je protokol Obroč z žetonom (angl. *token ring*), pri katerem so vozlišča povezana v obroč. Pri tem protokolu že sama topologija omrežja (obroč) določa vrstni red potovanja žetona med vozlišči, žeton torej kroži po obroču. Če želi vozlišče poslati okvir, lahko to naredi takrat, kadar ima žeton. Okvir običajno pošlje po obroču v isti smeri, kot po njemu kroži žeton. Nadaljnja vozlišča okvir posredujejo naprej do naslovnika in tudi naslovnik okvir pošlje naprej po obroču. Ko se okvir vrne do pošiljatelja, ga le-ta odstrani z obroča ter poda žeton naprej. Med prenosom okvirja se tako žeton zadržuje pri pošiljatelju. S tem se preprečujejo trki, saj lahko oddaja le vozlišče, ki ima žeton. Takšno obliko protokola, ki uporablja žetone, uporabljajo različni protokoli na povezavni plasti, kot so Token Ring⁴ (IEEE 802.5), FDDI (Fiber Distributed Data Interface, ANSI X3T12) in RPR (Resilient Packet Ring, IEEE 802.17). Slabost rešitev s podajanjem žetnov je, da so vse vpletene naprave hkrati tudi šibke točke: če katerakoli odpove, se veriga prekine in komunikacija ni več možna.

Prednost protokolov za izmenični dostop je v tem, da lahko vsako vozlišče pri pošiljanju izkoristi maksimalno kapaciteto kanala. Če odmislimo porabljen čas za režijo (poizvedovalno sporočilo ali žeton), lahko tudi rečemo, da so protokoli iz te družine pravični, saj skupno kapaciteto kanala enakomerno razdelijo med udeležence. V realnosti ne smemo pozabiti, da zaradi režije učinkovitost ni stoodstotna.

3.1.4 Ostale storitve povezavne plasti

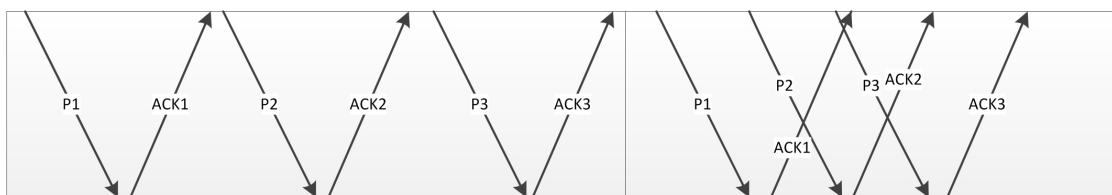
V razdelku 3.1 smo omenili, da povezavna plast lahko skrbi tudi za zagotavljanje zanesljive dostave in kontrolo pretoka (ne pa vedno in pri vsakem protokolu). Ker bomo te storitve zelo natančno opisali v poglavju o transportni plasti (poglavlje 5), jih v tem razdelku opisemo zgolj na kratko.

⁴pisano z velikama začetnicama

Zagotavljanje zanesljive dostave

Zagotavljanje zanesljive dostave pomeni, da imata pošiljatelj in prejemnik implementirano neko vrsto potrjevanja dostave okvirja (pozitivna potrditev) in nek način zahteve za ponovni prenos okvirja (negativna potrditev). To ne pomeni, da se vsak okvir zagotovo pravilno prenese na drugo stran, ampak da lahko prenos ponavljata, dokler ni uspešen.

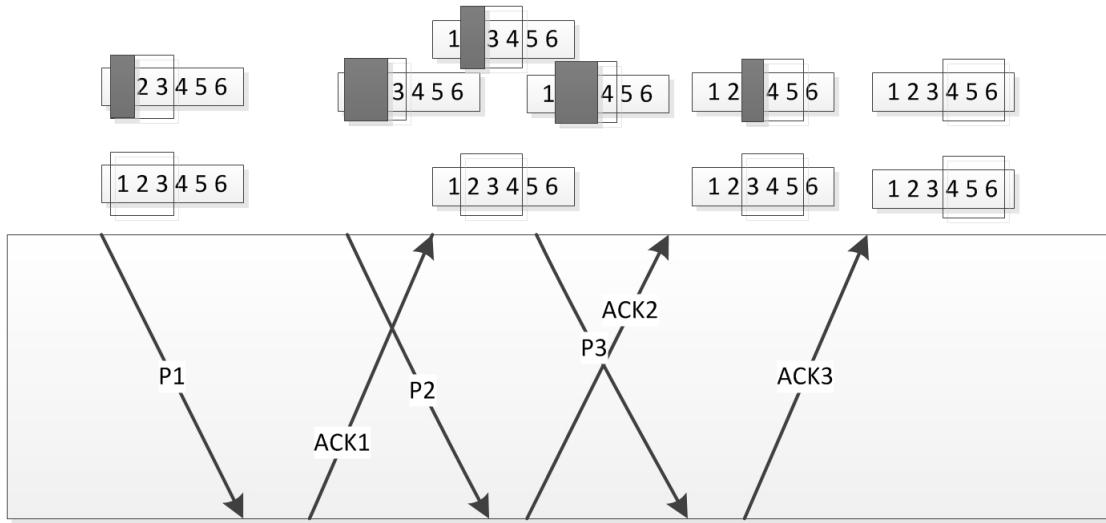
Glede na dinamiko oddajanja okvirjev ločimo dva načina potrjevanja: sprotno potrjevanje in potrjevanje s tekočim pošiljanjem. Pri sprotnem potrjevanju pošiljatelj z oddajo naslednjega okvirja čaka, dokler ne dobi potrditve pravkar oddanega okvirja. Pri tekočem pošiljanju pa to ni potrebno, okvirje oddaja enega za drugim, potrditev pa pride kasneje, ko je bilo morda oddanih že več naslednjih okvirjev. Oba načina prikazuje slika 3.9. Sprotno potrjevanje je dokaj neučinkovito, saj ni mogoče izkoristiti polne hitrosti prenosnega medija. Zlasti se to kaže na povezavah, ki imajo veliko latenco. Zato se danes mnogo bolj uporablja tekoče pošiljanje.



Slika 3.9 Levo: sprotno potrjevanje (preden se odda naslednji okvir, pošiljatelj počaka na potrditev prejšnjega). Desno: tekoče pošiljanje (okvirji se tekoče pošiljajo, ne da bi za vsak okvir pred oddajo čakali na prejšnje potrditev; pošiljanje se začasno prekine, če je oddanih in nepotrjenih toliko okvirjev, kot določa širina drsečega okna).

Pri tekočem pošiljanju ni dovoljeno oddajati okvirjev kar v nedogled, če potrditev ni od nikoder. Pač pa je število oddanih okvirjev navzgor omejeno. Omejitev imenujemo "širina drsečega okna" in opredeljuje največjo količino podatkov, ki jih lahko pošiljatelj odda, ne da bi dobil kako potrditev prejema. Pojasnimo, kaj pomeni izraz *drseče okno*. Okno si lahko predstavljamo kot shrambo okvirjev, v kateri so vsi okvirji, ki so že bili oddani, niso pa bili še potrjeni, poleg tega pa toliko naslednjih še neoddanih okvirjev, kolikor je še prostora. Okno tako z vsako prejeto potrditvijo "drsi" po zaporedju okvirjev, ki so namenjeni pošiljanju. Potrjen paket izpade iz okna, v okno pa pride naslednji paket, ki je pripravljen za pošiljanje. Torej s prejemom potrditve okno zdrsne naprej za en okvir. Pojasnimo še pojem *odpiranja* in *zapiranja okna*. Z vsako oddajo okvirja se okno zapre za eno režo. To pomeni, da imamo znotraj okna na voljo en okvir manj za pošiljanje. Ko smo oddali vse okvirje iz okna, je okno zaprto: to pomeni, da ne moremo ničesar več oddati, dokler ne začnejo prihajati potrditve in okno zdrsne naprej. S prejemom potrditve okno zdrsne naprej in se odpre za eno režo,

kar pomeni, da pošiljatelj lahko odda naslednji okvir. Prikaz primera drsenja in odpiranja okna je na sliki 3.10. Sprotno potrjevanje uporablja drseče okno dolžine 1 okvir.



Slika 3.10 Prikaz drsenja ter odpiranja in zapiranja okna pri tekočem pošiljanju, če je širina okna enaka 3 okvirje. Nad sliko vidimo v spodnji vrsti drsenje okna, v zgornji vrsti pa zapiranje in odpiranje okna.

Glede na vrsto potrditev ločimo neposredno in posredno potrjevanje. Pri neposrednem potrjevanju lahko prejemnik pošilja pozitivne in negativne potrditve. Pozitivna potrditev pomeni, da je bil okvir pravilno prenesen, negativna pa, da je prišlo do napak in da je potrebna ponovitev prenosa. Pri posrednem potrjevanju pa prejemnik pošilja le pozitivne potrditve. Če pri prejemu okvirja ugotovi, da je prišlo do napak, potrditev preprosto ne pošlje. V tem primeru pošiljatelj zazna napako, če potrditev še po dovolj dolgem času ne prispe ali pa če prispejo naslednje potrditve, medtem ko prejšnja manjka. V vsakem primeru pošiljatelj takoj pošlje manjkajoči okvir – čim ugotovi, da je prišlo do napake.

Glede na obnašanje pošiljatelja po napaki ločimo protokole na tiste s ponavljanjem zaporedja in tiste brez ponavljanja zaporedja. Pri protokolih s ponavljanjem zaporedja pošiljatelj po napaki ponovno pošlje okvir, zatem pa ponovno pošlje še vse okvirje, ki so bili poslani za njim. Ponovi torej celotno že oddano zaporedje od napake dalje. To je smiselno, če so na prenosnih medijih pogoste napake in je verjetno, da je bilo okvarjeno celotno zaporedje. Pri protokolih brez ponavljanja zaporedja pa pošiljatelj ponovno pošlje le okvarjeni okvir in ne ponavlja ostalih, ki so bili oddani za njim. V tem primeru mora prejemnik znati urediti okvirje glede na njihove zaporedne številke. Tak način je smiseln, če na prenosnem mediju le redko pride do motenj in napak, motnje pa so kratke in ni verjetno, da je okvarjenih več okvirjev zapored.

ARQ (angl. *Automatic Repeat Request*) ali **PAR** (angl. *Positive Acknowledgement with Retransmission*) je družina protokolov s posrednim potrjevanjem, torej samo s pozitivnimi potrditvami. Če po preteku določenega časa ne pride do potrditve, se okvir odda ponovno. Poznamo več različic. **Stop-and-Wait ARQ** predstavlja sprotno potrjevanje: pošiljatelj se po oddaji okvirja ustavi in čaka na potrditev. Ko potrditev prispe, odda naslednji okvir. Če potrditve določen čas ni, odda isti okvir ponovno. **Go-Back-N ARQ** predstavlja tekoče pošiljanje s ponavljanjem zaporedja. Če potrditev N-tega okvirja po preteku določenega časa ne pride, pošiljatelj ponovi pošiljanje vseh N zadnjih okvirjev. **Selective-Repeat ARQ** predstavlja tekoče pošiljanja brez ponavljanja zaporedja: če potrditev N po določenem času ne pride, pošiljatelj ponovno odda samo N-ti okvir, nato pa nadaljuje od tam naprej, kjer je končal.

Go-back-N ARQ se uporablja v protokolu HDLC (angl. *High-Level Data Link Control*) organizacije ISO in številnih izpeljankah. Selective-Repeat ARQ pa se rahlo obogaten uporablja v protokolu TCP na transportni plasti.

Nadzor pretoka

Nadzor pretoka pomeni funkcionalnost protokola, ki omogoča prejemniku okvirjev, da začasno ustavi ali upočasni pošiljatelja, če ta okvirje pošilja prehitro.

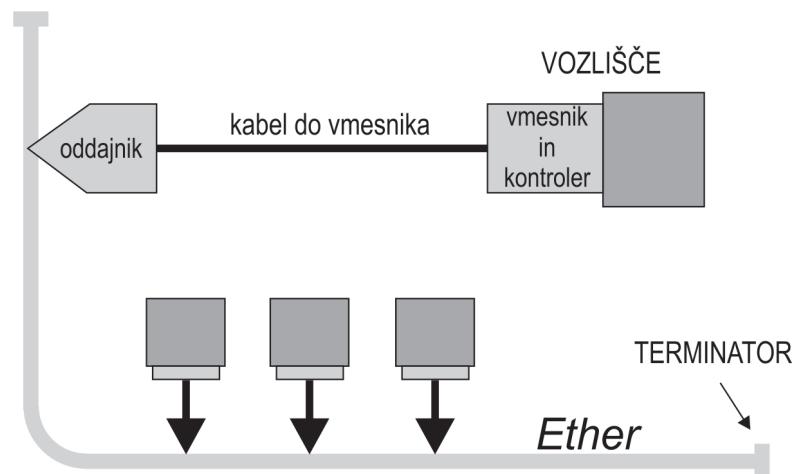
XON/XOFF. Najpreprostejši način nadzora pretoka je tak, da preobremenjeni prejemnik v določenem trenutku pošiljatelja preprosto ustavi – pošlje sporočilo “prenehaj pošiljati” - transmit off ali XOFF. Ko spet lahko sprejema, obvesti pošiljatelja s sporočilom “nadaljuj s pošiljanjem” - transmit on ali XON. Tak način se je uporabljal na primer na že dolgo zastarelih napravah – teleprinterjih ali pa pri pošiljanju podatkov na počasne tiskalnike. Sporočili sta bili predstavljeni s posebnima ASCII znakoma DC1 in DC3 in sta se kar vrinili med podatka. Protokol RS-232 pa deluje podobno, le da uporablja sporočili RTS (request to send) in CTS (clear to send) – zahteva za pošiljanje in dovoljenje za pošiljanje, ki ju pošilja po ločenem kontrolnem kanalu.

Drseče okno. Pri mehanizmih z drsečim oknom ob ustreznih velikosti okna ne potrebujemo dodatnega nadzora pretoka, saj lahko pošiljatelj pošlje le toliko okvirjev, kolikor mu dovoljuje širina okna, potem pa mora čakati na prejemnikove potrditve. Z dinamiko oddajanja potrditev lahko torej prejemnik dovolj natančno uravnava intenzivnost pošiljanja.

3.2 Pomembni protokoli na povezavni plasti

3.2.1 Ethernet

Ethernet (IEEE 802.3) je zagotovo eden najbolj razširjenih protokolov na povezavni plasti, njegova zgodovina pa sega še v sedemdeseta leta. V tistem času se je snovalec Etherneta, Robert Metcalfe, domisliil ideje, da bi deljeni komunikacijski kanal preprosto implementiral z bakrenim medijem, na katerega bi istočasno priključil več naprav. Ker je medij poimenoval z imenom *ether*, je komunikacijski protokol dobil ime *Ether–net*, kot je prikazano na sliki 3.11.



Slika 3.11 Konceptualna zasnova Etherneta, ki jo je zasnoval Robert Metcalfe.

Ethernet je zasnovan kot nepovezavna storitev, kar pomeni, da vmesniki ne izvajajo faze vzpostavljanja in rušenja povezav, temveč okvirje le oddajo na medij. Zaradi možnosti trka ovirjev uporablja Ethernet protokol CSMA/CD, ki smo ga omenili v razdelku 3.1.3. Ethernet implemenitra še dva specifična dodatka:

1. **Eksponentno povečevanje časa čakanja po zaporednih trkih** (angl. *exponential backoff time*): V primeru, da se zgodi trk ovirjev dveh vozlišč, protokol CSMA/CD veleva, da mora vsako izmed vozlišč počakati naključen časovni interval pred poskusom naslednje oddaje. Če se trk ovirjev večkrat ponovi, specifikacija določa, da lahko vsako vozlišče v vsaki iteraciji trka izbira svoj čakalni čas z daljšega razpoložljivega časovnega intervala. Ta interval se med iteracijami eksponentno povečuje (množi z 2), določen pa je kot čas, v katerem se lahko prenese določeno število ovirjev. Primer:

- ob prvem trku ovirjev vozlišči naključno izbereta, ali pošljeta takoj (po 0 vmesnih poslanih ovirjih) ali po 1 izpuščenem ovirju,

- ob drugem trku istih okvirjev vozlišči naključno izbereta, ali pošljeta po 0, 1, 2 ali 3 izpuščenih okvirjih,
- ob tretjem trku vozlišči naključno izpustita 0-7 okvirjev pred ponovnim pošiljanjem,
- itd.

Vidimo lahko, da s časom verjetnost ponovnega trka med okvirji pada in da se s tem povečuje verjetnost uspešnega prenosa. Najdaljši čas čakanja je tudi navzgor omejen, v standardu IEEE 802.3 na primer znaša čas prenosa 1023 okvirjev.

2. **Motilni signal ob zaznavi trkov** (angl. *jam signal*): Ko Ethernet vozlišča zaznajo trk, pošljejo drugim vozliščem *motilni signal*, ki je dolg 32 bitov in sestavljen iz izmenjujočih enic in ničel. Namen tega signala je, da vozlišče okvari del okvirja, v katerem se nahajajo biti za preverjanje pravilnosti (CRC) in s tem nedvoumno zagotovijo, da druga vozlišča po pomoti ne sprejmejo okvarjenega okvirja kot pravilnega.

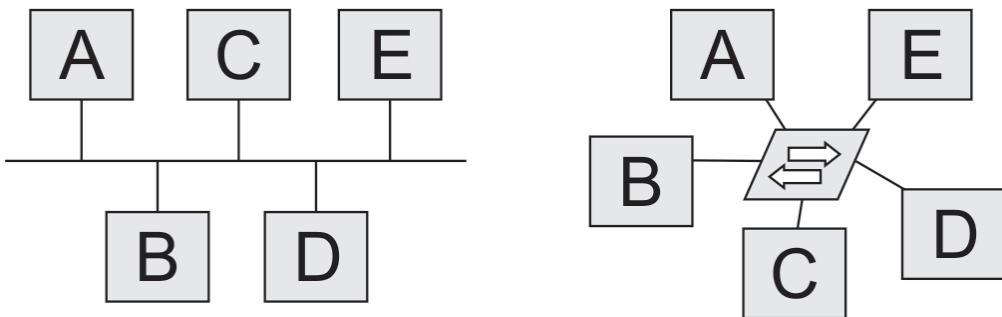
Z obema specifičnima dodatkoma dosega Ethernet učinkovitost rabe medija v razponu med 85% in 100%.

Topologije omrežja Ethernet

Po Metcalfeovi idejni zasnovi se je Ethernet začel naglo razvijati. Njegova zgodnja fizična implementacija, ki je prikazana na sliki 3.11, je bila skoraj dobesedna pretvorba skice v realnost: za povezovanje vozlišč je bil uporabljen koaksialni kabel, na katerega so bila priključena vozlišča z vmesnimi razdelilci tega kabla, imenovanimi T-členi. Na oba konca kabla je bilo potrebno namestiti element, imenovan *terminator* (50Ω upor), ki je preprečeval odboj signala od konca medija, saj bi ta povzročil trk signala s samim seboj.

Topologijo omrežja, ki smo jo ravnokar opisali, imenujemo topologija **vodila** (angl. *bus*) (slika 3.12 - levo), ta pa ima dve veliki pomanjkljivosti. Prva je ta, da pri oddajanju okvirja, le-tega "slišijo" vsa druga vozlišča in če prične oddajati tudi katerokoli izmed njih, lahko povzroči trk (temu pravimo, da so vsa vozlišča v *isti kolizijski domen*). Druga velika pomanjkljivost je ta, da ima topologija vodila izrazito enotno točko odpovedi: če se kabel v poljubni točki med vozlišči prekine, se s tem onemogoči komunikacija v celotnem sistemu.

Kot rešitev omenjenih šibkosti topologije vodila je razvoj topologije omrežja Ethernet okoli leta 2000 začel voditi k topologiji **zvezda**, ki je prikazana na sliki 3.12 - desno. Pri tej topologiji so vozlišča priključena na centralno napravo, ki je lahko bodisi **zvezdišče** (ali *razdelilec*, angl. *hub*) ali **omrežno stikalo** (angl. *switch*). Takšna topologija zvišuje robustnost omrežja, saj v primeru odpovedi ene povezave lahko ostala vozlišča še vedno komunicirajo. Šibka točka pa še vedno ostaja centralna naprava. Če je centralna naprava omrežno stikalo,



Slika 3.12 Topologija vodila (levo) in topologija zvezda (desno)

lahko le-to deli omrežje na več različnih kolizijskih domen, kar je še dodatna prednost, o kateri bomo podrobneje govorili v razdelku 3.2.1.

Različice EtherNETA

Ker je implementacija povezavnega protokola Ethernet pogojena s fizikalnimi lastnostmi prenosnega medija, obstaja glede na uporabljen fizični medij in različne standarde hitrosti več njegovih različic. Različice EtherNETA označujemo z oznakami oblike *hitrost*BASE-*medij*, kjer *hitrost* podaja hitrost prenosa po mediju (glej tabelo 3.1), kratica *BASE* poudarja, da signal zavzema cel razpoložljiv frekvenčni spekter (angl. *baseband*), *medij* pa podaja dodatno oznako, ki opisuje lastnost medija (glej tabelo 3.2).

oznaka	hitrost
10	10 Mbps
100	100 Mbps
1000	1000 Mbps
1G	1 Gbps
40G	40 Gbps
100G	100 Gbps

Tabela 3.1 Razlage oznak hitrosti v različicah EtherNETA

Z uporabo navedenih kratic lahko pogoste različice EtherNETA zapišemo npr. kot 100BASE-TX (100 Mbps Ethernet, ki uporablja zvito bakreno parico) ali pa 1000BASE-LX (1000 Mbps Ethernet, ki uporablja optično vlakno). Kljub razlikam pa imajo vse od njih isto obliko okvirja in uporabljuje isti protokol za dostop do medija.

oznaka	lastnost medija
2 ali 5	koaksialni medij največje dolžine 200 m ali 500 m (zastarela oznaka za topologijo vodila)
T	uporaba zvite bakrene parice (T=twisted)
FX	uporaba para optičnih vlaken FDDI (F=fiber), kratkih valov (850 nm) in sistema za kodiranje signala imenovanega <i>external sourced coding</i> (X=eXternal)
SX	uporaba dveh optičnih vlaken, kratkih (S=short) valov (850 nm) in sistema <i>external sourced coding</i> (X=eXternal)
LX	uporaba optičnega vlakna in dolgih (L=long) valov (1310 nm) in sistema <i>external sourced coding</i>
SR	uporaba dveh optičnih vlaken, kratkih (S=short) valov (850 nm) in sistema <i>scrambled encoding</i> (R=scRambled)
LR	uporaba dveh optičnih vlaken, dolgih (L=long) valov (1310 nm) in sistema <i>scrambled encoding</i> (R=scRambled)
ER	uporaba dveh optičnih vlaken, zelo dolgih (E=extra long) valov (1550 nm) in sistema <i>scrambled encoding</i> (R=scRambled)

Tabela 3.2 Razlage oznak medija v različicah Etherneta

Naslavljanje vozlišč

Ker Ethernet uporablja deljen komunikacijski medij, to pomeni, da lahko vozlišča slišijo tudi okvirje, ki niso namenjeni njim. Za opredelitev, komu je posamezen okvir namenjen, moramo torej uvesti *sistem naslavljanja* vozlišč, tako da vsaka naprava lahko za vsak okvir ugotovi, komu je namenjen. Če je namenjen njej, ga bo prebrala, dekapsulirala in vsebino predala višji plasti v komunikacijskem skladu. Okvire, ki niso namenjeni njej, pa bo zavrgla.

Za označevanje vozlišč uporablja Ethernet strojne ali MAC (angl. *Media Access Control*)⁵ naslove. Če smo natančni, naslove MAC pravzaprav uporablja cela družina protokolov IEEE 802.x, kamor spada tudi Ethernet, ki nosi oznako 802.3. Vsaka naprava ima svoj naslov MAC, ki je načeloma enoličen⁶ in ga določi že proizvajalec vmesnika ter je nespremenljivo zapisan v strojni opremi omrežnega vmesnika.

Fizični naslovi ali naslovi MAC so dolžine 48 bitov (6 bajtov), običajno jih zapišemo kot 12 šestnajstih znakov, ločenih z vezaji (ali dvopičji). Prva polovica naslova MAC (24 bitov) je identifikator proizvajalca (angl. *Organisationally Unique Identifier (OUI)*), ki ga proizvajalec zakupi od organizacije IEEE⁷. Druga

⁵Razlikujmo MAC naslove od MAC protokolov za dostop do skupinskega medija!

⁶V praksi je naslovov MAC premalo za vse naprave na svetu, zato se isti naslovi uporabljajo večkrat. Torej je teoretično možno, da zaradi tega pride do trka. Ker pa se naprave naslavljajo z naslovi MAC le znotraj posameznega omrežja, je zelo malo verjetno, da se to res zgodi.

⁷Evidenca je na naslovu <http://standards.ieee.org/develop/regauth/oui/oui.txt>.

★ Primer 3.5: Naslov

01011100 01100011 10111111 10010000 01110101 10110001

lahko ob upoštevanju $5_{16} = 0101_2$, $C_{16} = 1100_2$ itd. zapišemo kot:

5C-63-BF-90-75-B1.

Prva polovica naslova (5C-63-BF) določa proizvajalca kartice, ki je v tem primeru TP-LINK technologies, 90-75-B1 pa je specifična oznaka posameznega vmesnika, določena s strani proizvajalca.

polovica (24 bitov) predstavlja naslov omrežnega vmesnika, ki ga na enoličen način določi proizvajalec. V vsakem zakupljenem naboru naslovov ima torej proizvajalec svobodo določiti 2^{24} različnih naslovov omrežnih vmesnikov.

Kot poseben naslov omenimo naslov FF-FF-FF-FF-FF-FF (48 enic), ki ga imenujemo *naslov za razpošiljanje* (angl. *broadcast address*). Okvir, ki je naslovljen na ta naslov, je namenjen vsem vozliščem, ki so priključena na medij.

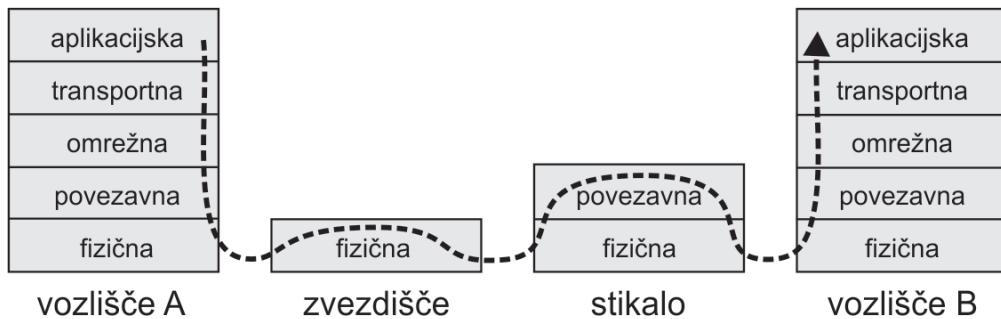
Omrežna Ethernet stikala

Omrežna stikala so naprave, ki smo jih omenili že pri opisu omrežne topologije zvezda v razdelku 3.2.1. So centralne omrežne naprave, na katere lahko priključimo več naprav, in na ta način nam omogočajo njihovo povezovanje v isto krajevno (lokalno) omrežje. Za fizični priklop naprav imajo stikala več vrat (angl. *port*), na katere priključimo različne "krake" omrežja, v katerem je stikalo v sredini. Na vrata omrežnega stikala lahko priključimo tudi druga omrežna stikala in na ta način zgradimo hierarhično topologijo krajevnega omrežja.

Preden se lotimo podrobnejšega opisovanja funkcije stikal, omenimo najprej še preprostejšo omrežno napravo, ki nam ravno tako omogoča priključitev več naprav v isto omrežje – to je *zvezdišče* ali *omrežni razdelilec* (angl. *hub*). Tudi zvezdišče ima več vrat, na katera lahko priključimo različne naprave, njegova naloga pa je preprosta: vsak signal, ki prispe na določena vrata, mora razmnožiti na vsa preostala vrata. Na ta način zvezdišče oponaša komunikacijo, ki jo poznamo s topologije vodila, kjer lahko vsa vozlišča slišijo poslane okvirje vseh drugih vozlišč. Na tem mestu poudarimo, da zvezdišče dela z *električnim signalom*, ki ga razdeli in ojača, ne zaveda pa se oblike okvira, ki je predstavljen s tem signalom. Zato pravimo, da zvezdišče deluje na najnižji, fizični plasti, kot to prikazuje tudi slika 3.13. Slabost zvezdišča je, da so vsa priključena vozlišča v isti kolizijski domeni in če generirajo veliko omrežnega prometa, je tudi trkov tako veliko, da postane omrežje neuporabno počasno. Zato gledamo na zvezdišča kot zastarela in jih v krajevnih omrežjih le še redko srečamo.

V primerjavi z zvezdišči opravljajo stikala bolj zahtevne naloge. Namesto preprostega razmnoževanjega signala stikala izvajajo *preklapljanje* (angl. *switching*, od tod tudi ime naprave) med pari vrat, hranijo podatke o priključenih

napravah in razumejo strukturo okvirjev. Ker uporabljajo podatke iz glave okvirja in so osnovne enote, s katerimi delajo, okvirji, pravimo da stikala delujejo na povezavni plasti, kot je tudi prikazano na sliki 3.13.



Slika 3.13 Potek omrežne komunikacije skozi zvezdišče in omrežno stikalo: zvezdišče deluje na fizični plasti komunikacijskega modela, stikalo pa na povezavni, kar pomeni, da razume strukturo okvirja.

Osnovne funkcije in prednosti stikala lahko strnemo v naslednje točke:

1. Zaradi preklapljanja med pari vrat stikalo omogoča **več hkratnih neodvisnih povezav** med vozlišči. Dve povezavi sta neodvisni, če ni galvanske povezave med njima, kar pomeni, da se signala na teh dveh povezavah medsebojno ne motita in ne povzročata medsebojnih kolizij. Tej prednosti stikala pravimo, da stikalo razbije omrežje na *manjše ločene kolizijske domene*.
2. Omogočena je **istočasna dvosmerna povezava** med vozlišči.
3. Na vsaka vrata lahko priključimo **drugo različico protokola Ethernet** (glej različice v razdelku 3.2.1) oziroma različno hitro povezavo.
4. Stikalo v sebi hrani **stikalno tabelo**, v kateri si skozi čas beleži, na katerih vratih se nahaja kateri naslov MAC. Stikalna tabela hrani trojice podatkov:

naslov MAC	vrata	TTL
------------	-------	-----

Vsak zapis je opremljen tudi z vrednostjo TTL (angl. *Time-To-Live*), ki določa čas veljavnosti zapisa (ne želimo, da je podatek trajen, saj lahko naprave odklapljam, selimo in priklapljam nove. Stikalo mora tem sprembam slediti in imeti možnost, da podatke v tabeli osvežuje).

Delovanje stikal je torej transparentno uporabniku in za vzpostavitev povezljivosti med vozlišči ne potrebuje posegov administratorja. Zato lahko rečemo tudi, da deluje po principu samodejne nastavitev (angl. *plug-and-play*). Najbolj izrazita prednost stikala je, da se s sprejemanjem okvirjev sam nauči, na katerih vratih se nahajajo kateri pošiljatelji. Ob vsakem prejemu si zapomni, na katerih vratih je dobil okvir, iz glave okvirja prebere naslov MAC pošiljatelja okvirja, potem pa

to zapiše v stikalno tabelo. Ob posredovanju okvirja lahko nato na podlagi stanja v stikalni tabeli izvede tri različna dejanja:

1. **POPLAVLJANJE** (angl. *flood*) okvirja na vsa vrata: do dejanja pride, kadar stikalo v stikalni tabeli nima zapisa, na katerih vratih se nahaja prejemnik okvirja,
2. **POSREDOVANJE** (angl. *forward*) okvirja na izbrana vrata: kadar ima stikalo v stikalni tabeli zapis, kje se nahaja prejemnik okvirja, posreduje okvir samo na njegova vrata,
3. **FILTRIRANJE** (angl. *filtering*): če je prejeti okvir namenjen prejemniku, ki je priključen na ista vrata kot pošiljatelj, stikalo okvir filtrira (oz. zavrže). V tem primeru je namreč prejemnik okvir že sprejel.

Poizvedovanje po fizičnih naslovih - protokol ARP

Na tem mestu se bralec lahko (upravičeno) vpraša, zakaj govorimo o naslovih MAC, ko pa se vendar komuniciranje v Internetu izvaja z uporabo vsesplošno znanih naslovov IP (*Internet Protocol*). Pojasnimo: Naslovi IP se ravno tako uporabljajo kot naslovi naprav, priključenih v omrežje, vendar se uporabljajo na višji, *omrežni* plasti, o njih bomo podrobneje govorili v poglavju 4. Za razliko od naslovov MAC, ki jim pravimo, da so fizični, so naslovi IP organizirani *logično*, tako da grupirajo skupine naslovov v manjše skupine in podskupine.

Opis odnosa med naslovi MAC in IP kliče po mehanizmu, s katerim lahko omrežna naprava na podlagi podanega naslova IP pridobi fizični naslov naprave, ki ji ta naslov IP pripada. To storitev nam omogoča protokol ARP (angl. *Address Resolution Protocol*), ki ga v komunikacijskem modelu uvrščamo med povezavno in omrežno plast, saj igra vlogo povezovalca med obema.

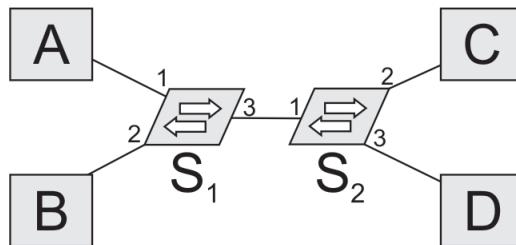
Vsako omrežno vozlišče hrani tabelo ARP, ki vsebuje zapise trojic:

naslov IP	naslov MAC	TTL
-----------	------------	-----

TTL (angl. *Time-To-Live*) predstavlja čas veljavnosti zapisa, ki se po pretečenem intervalu odstrani iz tabele, saj se lahko v omrežje čez čas priključi tudi kakšno drugo vozlišče z istim naslovom IP in drugim fizičnim naslovom MAC.

Zapis v tabeli ARP pomeni: naprava s fizičnim naslovom "naslov MAC" trenutno uporablja omrežni naslov "naslov IP". Kadar mora naprava poslati promet na določen naslov IP, potrebuje podatek o naslovu MAC naslednje naprave na poti do cilja, saj ji bo treba pošiljati okvirje. Če v tabeli ARP ne najde naslova MAC za ta naslov IP, sproži poizvedbo ARP, s katero povpraša po svojem krajevnem omrežju, kdo ima ta naslov IP. Naprava, ki ji ta naslov pripada, odgovori z odgovorom ARP, ki vsebuje tudi njen naslov MAC. Naprava, ki je poizvedovala, podatke iz prejetih odgovorov ARP sproti zapisuje v tabelo ARP. Protokol ARP torej poteka po naslednjih korakih:

★ **Primer 3.6:** Kot primer, kako stikala opravljajo zgoraj naštete funkcije, si poglejmo komunikacijo štirih vozlišč, ki je prikazana na naslednji sliki:



Denimo, da se v tem omrežju zgodijo dogodki, ki so prikazani v prvem stolpcu tabele na dnu tega primera (dogodki predstavljajo komunikacijo med različnimi pari vozlišč in iztek veljavnosti zapisa v stikalni tabeli). Tabela v preostalih stolcih prikazuje, kako se z vsakim posameznim dogodkom spreminja vsebina stikalnih tabel in katero dejanje izvede vsako izmed stikal ob dogodku.

Vidimo, da obe stikali poplavita prvi okvir, ki ga A pošlje C, ker sta obe stikalni tabeli na začetku prazni in nobeno od stikal ne more vedeti, na katerih vratih se nahaja ciljna naprava. Ob tej komunikaciji si obe stikali shranita številko vrat, na katera je priključen A. Pri drugem dogodku, ko C odgovori A, obe stikali že vesta, na katera vrata je priključena naprava A, zato lahko obe stikali okvir posredujeta na prava vrata in si pri tem dodatno zapomnita tudi lokacijo C. Ker imajo zapisi TTL, čez nekaj časa najstarejši zapis v stiku S_1 poteče in stikalna tabela se ustrezno spremeni. Ob zadnjem dogodku, okvirju, ki ga B nameni A, zato stikalo S_1 ponovno poplaví poslani okvir, stikalo S_2 pa ga filtrira (zavrije), saj še vedno ve, da se A nahaja na istih vratih kot okvir, ki je ravnokar prispel.

dogodek	stikalna tabela S_1 (naslov/vrata)	dejanje S_1	stikalna tabela S_2 (naslov/vrata)	dejanje S_2
$A \rightarrow C$	A/1	poplavi	A/1	poplavi
$C \rightarrow A$	A/1 C/3	posreduj na 1	A/1 C/2	posreduj na 1
potek najstarejšega zapisa v S_1	C/3	/	A/1 C/2	/
$B \rightarrow A$	C/3 B/2	poplavi	A/1 C/2 B/1	filtriraj

★ **Primer 3.7:** Pojasnimo odnos med naslovi MAC in naslovi IP z analogijo iz vsakdanjega življenja. Vsaka oseba ima svojo davčno številko, ki je zanj enolična in na ta način predstavlja ekvivalent njegovemu fizičnemu naslovu, saj je po njem možno osebo enolično identificirati. Istočasno ima vsaka oseba tudi svoj domači naslov, ki pa predstavlja ekvivalent IP (logičnega) naslova, saj lahko po tem podatku osebo najdemo. Čeprav je naslov prebivališča zamenjiv (oseba se namreč lahko preseli), je davčna številka osebe stalna.

1. A razpošlje paket - **poizvedbo ARP** - vsem vmesnikom: "Sem naprava A z naslovom IP "IP A" in naslovom MAC "MAC A". Iščem napravo z naslovom IP "IP B". Ta poizvedba vsebuje naslova MAC in IP vozlišča A in naslov IP vozlišča B, čigar fizični naslov A zanima.
2. Iz naslova IP vozlišča B, ki je zapisan v poizvedbi ARP, vozlišče B ugotovi, da A sprašuje po njegovem fizičnem naslovu.
3. B odgovori vmesniku A z **odgovorom ARP**, v katerega zapiše svoj naslov MAC: "Sem naprava B z naslovom IP "IP B", moj naslov MAC pa je "MAC B". Okvir z odgovorom B naslovi direktno na vozlišče A, saj pozna njegov naslov MAC iz okvirja s poizvedbo, ki ga je prejel v prvem koraku.
4. A prejme odgovor od vozlišča B in shrani podatek v svojo tabelo ARP.

Okvir Ethernet

Sedaj vemo dovolj, da se lahko posvetimo strukturi okvirja protokola Ethernet. Ta je prikazana na sliki 3.14, iz katere vidimo, da vsebuje polja, ki so razložena v tabeli 3.3.

Danes se največ uporablja mlajša specifikacija, imenovana Ethernet II, ki so jo razvili Dec, Intel in Xerox. Tu je format okvirja posodobljen in ne vsebuje več preambule, s čimer se zmanjša režija. Polje Dolžina pa postane TIP (Ethertype) in določa, kakšna vsebina se nahaja v okvirju kot podatkovno breme (na primer paket IPv4, paket IPv6, ARP, ipd.)

(PREAMBULA)	naslov PREJEMNIKA	naslov POŠILJATELJA	TIP/ dolžina	PODATKI	CRC
8 bajtov	6 bajtov	6 bajtov	2 bajta	46 - 1500 bajtov	4 bajti

Slika 3.14 Okvir protokola Ethernet/Ethernet II

3.2.2 WiFi – IEEE 802.11

Z besedama WiFi ali WLAN opisujemo celo družino standardov IEEE 802.11. Ti se med seboj razlikujejo tako po tehnologiji kodiranja signala, oddajni moči in dometu, kot tudi po uporabljenem frekvenčnem območju in doseženi hitrosti prenosa. Družina se še širi, saj se pojavljajo potrebe po še hitrejših prenosih. Z že zelo zasedenega 2.4 GHz in dokaj zasedenega 5 GHz območja, za uporabo katerih je potrebna licenca, se novejše različice selijo na zelo visoke ali zelo nizke nelicenčne frekvence (60 GHz za hitre prenose na kratke razdalje in 900 MHz za počasnejše prenose na daljše razdalje), kot je prikazano v tabeli 3.4.

POLJE	DOLŽINA	POMEN
(PREAMBULA)	8 bajtov	Zaporedje oblike $7x\ 10101010 + 1x\ 10101011$, ki je namenjeno usklajevanju signala oddajnika in prejemnikov. Pri usklajevanju se uskladita pošiljateljeva in prejemnikova ura, razbere se tudi pošiljateljeva hitrost prenosa. Celo zaporedje 8 bajtov je mišljeno kot opozorilo prejemniku, da v nadaljevanju prihajajo pomembni podatki, ki so glavni sestavni del okvirja.
NASLOV PREJEMNIKA	6 bajtov	Naslov MAC prejemnika. Vozlišče bo okvir sprejelo, če to polje vsebuje njegov naslov MAC ali pa naslov za razpošiljanje (FF-FF-FF-FF-FF-FF).
NASLOV POŠILJATELJA	6 bajtov	Naslov MAC pošiljatelja.
TIP/DOLŽINA	2 bajta	V Ethernet II: Vrsta enkapsuliranih podatkov, npr. 0800H za protokol IP ali 0806H za protokol ARP ⁸ . V IEEE 802.3: dolžina podatkov v bajtih.
PODATKI	46-1500 bajtov	Enkapsulirani podatki (npr. paket omrežne plasti). Parameter MTU (angl. <i>Maximum Transmission Unit</i> , ki je specifičen za vsak protokol na povezavni plasti, določa največjo dolžino podatkov, ki so lahko enkapsulirani v okvir).
CRC	4 bajti	Biti za preverjanje pravilnosti okvirja. Če se CRC ne ujema z vsebino okvirja, prejemnik okvir zavrne.

Tabela 3.3 Pomen polj v okvirju protokola Ethernet/Ethernet II

Različica	Frekvenčno območje	Hitrost	Opombe
802.11a	5, 6 GHz	54 Mb/s (tipično 23)	
802.11b	2,4, 5 GHz	11 Mb/s (tipično 4)	Domet do 140 m
802.11g	2,4, 5 GHz	54 MB/s (tipično 20)	Domet do 140 m
802.11n	2,4, 5 GHz	Do 600 Mb/s (150 Mb/s na posamezen tok)	MIMO – združevanje več vzporednih tokov
802.11ac	5 GHz	7-10 Gb/s (1.3 Gb/s na tok)	“Wave 2” - do 8 tokov
802.11ad	60 GHz (nelicencirano)	Do 7-10 Gb/s	Kratke razdalje (soba, npr. za HiFi komponente)
802.11ah	900 MHz (nelicencirano)	Do 40 Mb/s	Daljše razdalje, manjši pretok (npr. za senzorje)

Tabela 3.4 Razlage oznak hitrosti v različicah Etherne

Osnovni princip delovanja je preprost. Osnovno frekvenčno področje je razdeljeno v več kanalov in vsaka dostopna točka uporablja določen kanal. Ko naprava išče dostopno točko, pregleduje kanale in posluša, kje se oglaša naprava. Protokoli družine IEEE 802.11 delujejo po načelih CSMA/CA. Za izogibanje kolizijam se uporablja več različic, ki izhajajo iz že opisanega osnovnega principa, na primer MACAW (angl. *Multiple Collision Avoidance for Wireless*). Brezžična naprava tako pred oddajanjem s sporočilom RTS zaprosi za rezervacijo prenosnega kanala in šele po prejemu dovoljenja CTS odda podatke. Lahko se uporablja sprotno potrjevanje.

V specifikaciji se za dostopno točko uporablja izraz “basic service set”, s

kratico BSS. Vsaka dostopna točka ima svoj BSSID, 48-bitno številko, ki jo identificira na brezžičnem omrežju. Brezžična omrežja različnih dostopnih točk se lahko namreč med seboj fizično prekrivajo in brezžične naprave, ki so v dometu obeh, lahko sprejemajo broadcast sporočila od obeh. S pomočjo BSSID zaznajo, katera od teh sporočil oddaja "njihova" dostopna točka in jih sprejmejo ter katera oddaja sosednja točka - te zato zavržejo.

Načini vključevanja v brezžično omrežje

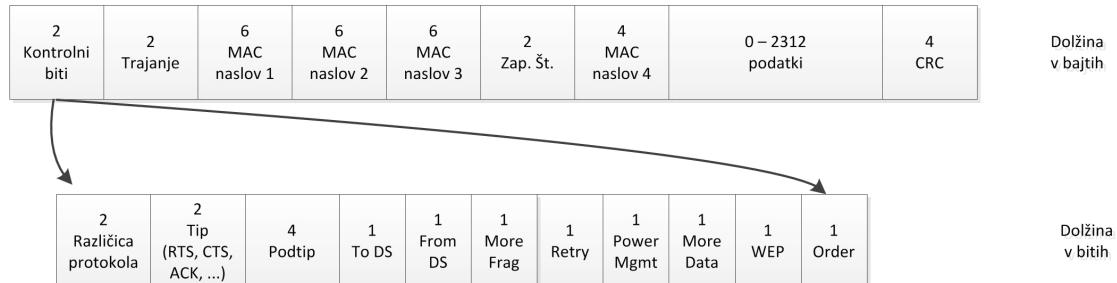
Za vključevanje v omrežje je namenjenih pet protokolarnih sporočil – različnih tipov okvirjev: *beacon*, *probe request*, *probe response*, *association request*, *association response*. Poznamo dva načina vključevanja:

- **Pasivno vključevanje:** dostopne točke periodično oglašujejo svojo prisotnost z okvirjem *beacon*, v katerem zapišejo svoje ime (SSID) in podprte hitrosti prenosa. Ko brezžična naprava sprejme tak okvir, se lahko odloči za vključitev v omrežje. V tem primeru odda dostopni točki zahtevo za pridružitev (angl. *association request*).
- **Aktivna izbira dostopne točke:** brezžični napravi, ki se želi vključiti v brezžično omrežje, ni potrebno čakati na sporočila *beacon*, saj lahko sama sproži iskanje dostopnih točk s sporočilom *probe* ("ali je tu kaka dostopna točka?"). Ko dostopna točka sprejme *probe*, nanj odgovori s sporočilom *probe response*, v katerem sporoči tudi svoje podatke. Brezžična naprava lahko nato odgovori z zahtevo za pridružitev – *association request*.

Tako pri pasivnem kot pri aktivnem pristopu dostopna točka nazadnje potrdi pridružitev v omrežje s potrditvijo pridružitve – *association response*.

Okvir 802.11

Format okvirja IEEE 802.11 (v času pisanja je še vedno enak za vse različice) prikazuje slika 3.15, polja pa so pojasnjena v tabeli 3.5. Povezavo med naslovi in kontrolnima bitoma *To DS* in *From DS*, ki sta odvisna od vrste in namena okvirjev, prikazuje tabelah 3.6 in 3.7.



Slika 3.15 Okvir protokola IEEE 802.11

3.2.3 Drugi protokoli iz družine IEEE 802

Protokolov ali tehnologij na povezavni plasti je mnogo in danes se tudi intenzivno razvijajo številne nove specifikacije. Omenimo le nekaj imen: Bluetooth, WiMax, ZigBee, ...

Specifikacija **Bluetooth** je že relativno stara, prvo različico je razvil Ericsson leta 1994. Določa komunikacijo na frekvenčnem področju 2.4 GHz, ki je razdeljeno na 79 kanalov, in s hitrostmi prenosa do 721 kb/s. Deluje na razdaljah manj kot 10 m, njegov prvotni namen pa je bil nadomestiti kable pri manj zahtevnih napravah, kot so miška, tipkovnica, slušalke in podobne. Bluetooth je bil nato standardiziran kot IEEE 802.15.1, ta standard pa se danes ne razvija več. Razvoj Bluetooth se nadaljuje pod okriljem samostojnega telesa za standardizacijo *Bluetooth Special Interest Group*, ki združuje več kot 25.000 podjetij, med katerimi kot promotorji nastopajo tudi Ericsson, Intel, Lenovo, Microsoft, Nokia in Toshiba.

Izhajajoč iz specifikacije za Bluetooth je IEEE oblikoval družino standardov IEEE 802.15, ki opredeljujejo komunikacijo v **osebnih omrežjih** (angl. *Personal Area Network, PAN*). Najbolj soroden je standard IEEE 802.15.3-2003, ki omogoča hitrosti prenosa od 11 do 55 Mb/s, njegova novejša različica 802.15.3-2009 pa hitrosti vse do 3 Gb/s.

IEEE 802.15.4-2003 (Low-rate WPAN) opredeljuje počasnejšo komunikacijo, kjer je pomembno zlasti varčevanje z energijo, saj naj bi brezžična naprava zdržala nekaj mesecev ali celo let. Na tem standardu temelji tudi specifikacija **ZigBee**: strojna oprema je poceni, prenosi imajo domet 10-100 m, vendar pa so se naprave sposobne organizirati v omrežje tipa *mesh* in pošiljati promet preko tega omrežja v več skokih. Hitrosti prenosa dosežejo 250 kb/s, kar je dovolj na primer za brezžična stikala (pametna hiša), senzorje, števce, semaforje in podobno. Na specifikaciji 802.15.4 temelji tudi 6LoWPAN (RFC 6282), ki je namenjen uporabi IPv6 v takšnih omrežjih.

Specifikacija IEEE 802.15.6-2012 opredeljuje komunikacijo v **telesnem omrežju** (angl. *Body Area Network, BAN*). Deluje na krajše razdalje, torej v bližnji okolici ali znotraj človekovega telesa, in z zelo majhnimi oddajnimi močmi. podpira hitrosti

Polje	Pomen
Trajanje / ID	To polje ima lahko več pomenov. Če je 15. bit enak 0, potem je vrednost polja število mikrosekund in pomeni predviden čas trenutnega prenosa. Če sta 14. in 15. bit enaka 1, gre za okvir <i>Poll</i> , ki ga odda sprejemnik, ko se zbudi iz varčevalnega načina. S tem okvirjem pove "svoj" dostopni točki, da zdaj spet posluša in se mu lahko pošlje morebitne čakajoče okvirje. Vrednost polja v tem primeru pove, za katero dostopno točko gre.
MAC naslovi 1, 2, 3, 4	Naslov pošiljatelja in končnega prejemnika, naslov vmesne dostopne točke in po potrebi še drugega posrednika. Kaj pomeni kateri naslov, je odvisno od vrednosti bitov <i>From DS</i> in <i>To DS</i> , kot opredeljuje spodnja tabela: <ol style="list-style-type: none"> 1. destination address - prejemnik, ki preda okvir na višje plasti 2. source address – pošiljatelj, ki je generiral okvir 3. receiver address – končna brezžična naprava (končni prejemnik ali pa dostopna točka, ki preda okvir v ozičeno omrežje) 4. transmitter address - naslov oddajnika pri uporabi brezžičnega mostu
Zap. št.	Zaporedna številka, ki omogoča razločevanje okvirjev med seboj in izločanje duplikatov (na primer če se je izgubila potrditev in je bil ponovno poslan okvir, ki je bil sicer že pravilno sprejet). Če gre za fragmentiran okvir, prvi štirje biti pomenijo zaporedno številko fragmenta.
Kontrolni biti:	<ul style="list-style-type: none"> • Različica: Katera od različic 802.11 se uporablja (trenutno imajo vse različice enako glavo, zato je vrednost vedno 00). • Tip in podtip: Skupaj opredeljujeta tip okvirja in njegovo funkcijo. Tipi so trije: 00 za upravljanje (beacon, association request in response, probe request in response), 01 za kontrolne okvirje (RTS, CTS, ACK, ...) in 10 za podatkovne okvirje (podatki, podatki + ACK, prazen okvir, poizvedba, ...). Tip 11 ni v uporabi. • To DS in From DS: Opredeljujeta, ali je okvir namenjen v omrežno hrbitenico (angl. <i>distribution system</i>) ali prihaja iz nje. Kombinacija obeh vrednosti tudi opredeljuje pomen vseh štirih naslovnih polj, kot kaže tabela 3.6. • More frag.: Če je veliko motenj, WiFi oddajnik razdrobi večje okvirje v več manjših, saj je bolj verjetno, da bo prenos uspešen, oziroma je manj ponavljanj. Če je ta bit postavljen, pomeni, da paketa še ni konec in da pridejo še okvirji s kosi trenutnega paketa. • Retry: Če je ta bit postavljen, gre za ponoven prenos že prej poslanega okvirja. • Power mgmt.: Če je ta bit postavljen, bo po trenutnem prenosu oddajnik preklopil na način varčevanja z energijo in postal neaktivен. • More data: Če je ta bit postavljen, ima pošiljatelj v vmesniku pripravljenih še več okvirjev za oddajo, ki bodo sledili trenutnemu. • WEP: Če je ta bit postavljen, je vsebina okvirja kriptirana z WEP. • Order: Če je ta bit postavljen, morajo biti paketi urejeni po vrstnem redu oddaje (npr. za VoIP).

Tabela 3.5 Pomen polj v glavi okvirja IEEE 802.11

prenosa do 10 Mb/s in deluje na frekvencah, ki so odobrene tudi z medicinskega vidika.

Čisto na drugem koncu pa je specifikacija IEEE 802.15.7 - Visible Light Communication (VLC) ali tudi **Li-Fi**, ki za komunikacijo uporablja frekvenčni spekter vidne svetlobe (valovne dolžine 380 do 780 nm). Namenjena je za brezžične LAN in MAN povezave, torej na daljše razdalje kot prejšnja specifikacija, in z zadostno

70 Poglavlje 3 Povezavna plast

	ToDS = 0	ToDS = 1
FromDS = 0	Vsi upravljalski (mgmt.) in kontrolni okvirji ter podatkovni okvirji znotraj brezžičnega omrežja posamezne dostopne točke (ne gredo v hrbtenico)	Podatkovni okvirji, ki jih odda brezžična naprava dostopni točki v ozičenem omrežju.
FromDS = 1	Podatkovni okvirji, ki so namenjeni brezžični dostopni točki nekje v ozičenem omrežju.	Podatkovni okvirji na brezžičnem mostu (bridge).

Tabela 3.6 Pomen kontrolnih bitov To DS in From DS

Funkcija	ToDS	FromDS	MAC1	MAC2	MAC3	MAC4
V območju ene dostopne točke	0	0	Končni prejemnik	Izvorni pošiljatelj	BSSID	–
Preko dostopne točke v omrežje	0	1	Končni prejemnik	BSSID	Izvorni pošiljatelj	–
Preko dostopne točke iz omrežja	1	0	BSSID	Izvorni pošiljatelj	Končni prejemnik	–
Brezžični most	1	1	Prejemnik z brezžičnega medija	Oddajnik na brezžični medij	Končni prejemnik	Izvorni pošiljatelj

Tabela 3.7 Interpretacija kontrolnih bitov ToDS in FromDS in njuna povezava z naslovi

hitrostjo prenosa za podporo večpredstavnih storitev. Trenutne implementacije zmorejo hitrosti okrog 1.6 Gb/s, predvidoma pa bodo kmalu dosegli okrog 5 Gb/s.

Številne tehnologije so razvila podjetja, ki niso želela čakati, ali bodo dolgorajni standardizacijski procesi obrodili uporabne sadove. Na področju pametnih hiš je danes široko uporabljan protokol Z-Wave, znani pa so še Osian, X10, Insteon, ... Na področju Interneta stvari se razvijajo novi, še bolj varčni in še bolj učinkoviti protokoli.

3.2.4 Point-to-point protocol (PPP)

Ker vemo, da obstaja več povezavnih protokolov, omenimo alternativo protokolu Ethernet, ki je po zasnovi zelo drugačna. Kot že ime protokola v naslovu tega razdelka pove, gre za protokol, namenjen povezovanju le dveh točk - vozlišč. Ta protokol se torej ne uporablja na deljenih medijih, temveč na dvotočkovnih povezavah, kjer imamo samo enega pošiljatelja in enega prejemnika. Iz tega razloga tudi ta protokol ne potrebuje posebnega protokola za dostop do fizičnega medija in naslavljanja vozlišč.

Okvir PPP

Slika 3.16 prikazuje, kako je strukturiran okvir protokola PPP. S slike lahko razberemo polja, ki so prikazana v tabeli 3.8 s podanimi razlagami pomena polj.

01111110 začetek	11111111 naslov	00000011 kontrola	PROTOKOL	PODATKI	CRC	01111110 konec
1 bajt	1 bajt	1 bajt	1-2 bajta	~1500 bajtov	2/4 bajti	1 bajt

Slika 3.16 Okvir protokola PPP

POLJE	DOLŽINA	POMEN
ZASTAVICA ZA ZAČETEK OKVIRJA	1 bajt	Ima vrednost 01111110 in igra podobno vlogo kot preambula pri protokolu Ethernet.
NASLOV (address) in KONTROLA (control)	1 + 1 bajt	Polji, ki hranita fiksni vrednosti 11111111 in 00000011 in sta bili namenjeni kasnejšim morebitnim razširitvam tega protokola, do katerih ni prišlo.
PROTOKOL	1 ali 2 bajta	Vrsta enkapsuliranih podatkov, podobno kot polje TIP pri protokolu Ethernet.
PODATKI	0 ali več bajtov	Enkapsulirani podatki (paket omrežne plasti), ki so spremenljive dolžine, prizveta dolžina je 1500 bajtov.
CRC	2 ali 4 bajti	16-bitna ali 32-bitna koda CRC za zaznavanje napak pri prenosu.
ZASTAVICA ZA KONEC OKVIRJA	1 bajt	Ima vrednost 01111110 in označuje konec okvirja, da prejemnik ve, kdaj lahko neha poslušati.

Tabela 3.8 Pomen polj v okvirju protokola PPP

Vrivanje zlogov

Protokol PPP je zanimiv, ker uporablja postopek vrivanja zlogov (bajtov). S tem postopkom protokol omogoča, da lahko enkapsulirani podatki hranijo poljubno zaporedje enic in ničel, tudi zaporedje 01111110, ki sicer predstavlja zastavico za začetek ali konec okvirja. V splošnem namreč velja, da če bi se omenjeno zaporedje pojavilo v enkapsuliranih podatkih, bi ga lahko prejemnik prezgodaj interpretiral kot konec okvirja, zaradi česar bi prišlo do napačne interpretacije vsebine celega okvirja.

Da pošiljatelj in prejemnik odpravita omenjeno težavo, uporabljata postopek vrivanja zlogov, ki veleva, da pošiljatelj pred zaporedje 01111110 vrine še en dodaten zlog (bajt), ki ima vrednost 01111101 in ga imenujemo *ubežna koda* (angl. *escape sequence*). Ko prejemnik prejme zaporedje zlogov 01111101 01111110 tedaj ve, da vrednost 01111110, ki sledi ubežni kodi ne pomeni konca okvirja. Zato ubežno kodo iz zaporedja odstrani, zaporedje 01111110 pa obravnavata kot del sprejetih podatkov. Na enak način se izvede vrivanje zlogov tudi v primeru, če poslani podatki vsebujejo zaporedje 01111101, ki sicer predstavlja ubežno kodo. V tem primeru pošiljatelj zapiše to zaporedje dvakrat, prejemnik pa upošteva le enega.

★ **Primer 3.8:** Denimo, da želi pošiljatelj prejemniku poslati zaporedje naslednjih treh zlogov: 10111001 01111110 00000010. Ker drugi zlog predstavlja zastavico za začetek/konec okvirja, bo pošiljatelj v okvir zapisal: 10111001 01111101 01111110 00000010. Prejemnik bo pri branju opazil, da je drugi zlog ubežna koda, zato jo bo odstranil in zaporedja in zlog, ki sledi (01111110), obravnaval kot del podatkov in ne zastavico za konec zaporedja. Prejeto zaporedje zlogov bo enako poslanemu: 10111001 01111110 00000010.

4 Omrežna plast

Omrežna plast se v komunikacijskem modelu nahaja nad povezavno plastjo in pod transportno (ali prenosno) plastjo. Ti pomeni, da nudi svoje storitve transportni plasti in zahteva izvedbo storitev od povezavne plasti, ki smo jo spoznali v poglavju 3. Pri postopku enkapsulacije se torej protokolarne enote transportne plasti, ki jih imenujemo tudi *segmenti*, enkapsulirajo v protokolarne enote omrežne plasti, ki jih imenujemo **paketi**. Protokol omrežne plasti je tisti, ki poskrbi, da se paket nato prenese od pošiljatelja do prejemnika. Za izvedbo tega prenosa se bo po komunikacijski poti uporabilo več povezav in s tem tudi več različnih povezavnih protokolov, ki bodo paket zaporedoma enkapsulirali v okvirje in ga dekapsulirali iz njih.

Pri potovanju skozi jedro omrežja paketi potujejo skozi naprave, imenovane **usmerjevalniki**. To so naprave, ki zagotavljajo povezljivost v jedru omrežja med različnimi povezavnimi protokoli in skrbijo za določanje poti, po kateri bo paket potoval skozi po omrežju. Za ta namen mora biti usmerjevalnik sposoben prebrati tiste podatke o pošiljatelju in prejemniku, ki se nahajajo v paketu, t. j. v protokolarni enoti omrežne plasti. Zato pravimo, da usmerjevalniki *delujo* na omrežni plasti.

Dve glavni nalogi usmerjevalnika sta:

- **posredovanje** (angl. *forwarding*): Ker usmerjevalniki predstavljajo “moste” med različnimi povezavami, morajo določati, na katera izhodna vrata morajo preposlati paket z vhodnih vrat – temu opravilu pravimo posredova-

★ **Primer 4.1:** V razdelku 3.1.1 smo podali primer, ki je predstavljal analogijo iz realnega sveta storitvam povezavne plasti – kako poiskati pot od Loma pod Storžičem do Kopra. Omenili smo, da so različni deli poti analogni različnim komunikacijskim povezavam, različna prevozna sredstva pa različnim povezavnim protokolom. V tem primeru nam je manjkal še *organizator poti*, ki je zadolžen za načrtovanje poti od Loma (pošiljatelj) do Kopra (prejemnika). Ta organizator so ravno protokoli omrežne plasti, ki so zadolženi za iskanje poti od pošiljatelja do prejemnika in izvedbo prenosa skozi omrežje. Če iščemo analogijo iz realnega sveta tudi za njih, bi jih z nekoliko šale lahko torej opisali kot potovalno agencijo.

nje. Ker že samo pregledovanje paketa zahteva svoj procesorski čas, mora usmerjevalnik poskrbeti, da v času procesiranja paketov ne bo prezrl drugih prihajajočih paketov. S tem namenom je vezje usmerjevalnika opremljeno z *vhodnim medpomnilnikom* (angl. *input buffer*), v katerem se hrani čakalna vrsta paketov, ki jih mora usmerjevalnik še obdelati in posredovati. Na podoben način imajo usmerjevalniki tudi *izhodni medpomnilnik* (angl. *output buffer*), kjer se hranijo paketi v vrsti za pošiljanje glede na kapaciteto in stanje povezave ali sprejemno hitrost prejemnika.

- **usmerjanje** (angl. *routing*): Je način določanja poti paketov od izvora do cilja. Te naloge ne izvaja en sam usmerjevalnik, ampak je skupinsko delo vseh usmerjevalnikov v omrežju, ki morajo med sabo izmenjevati podatke o možnih poteh. Za to nalogo skrbijo **usmerjevalni protokoli**.

V grobem lahko **naloge omrežne plasti** strnemo v tri skupine:

1. prenos paketov (pri katerem se v Internetu uporablja protokol IP),
2. izvedba signalizacije in prenos kontrolnih sporočil (v Internetu: protokol ICMP),
3. usmerjanje paketov z izbiro poti skozi omrežje (naloge usmerjevalnikov in usmerjevalnih protokolov).

4.1 Vrste omrežij

V splošnem ločimo dve skupini omrežnih protokolov in s tem tudi konceptualno drugačnih omrežij: to so **povezavna** in **nepovezavna** omrežja.

4.1.1 Povezavna omrežja

Povezavna omrežja ali *omrežja z navideznimi vodi* (angl. *virtual circuit*) so omrežja, ki izvirajo iz telefonije. Ker je prenos govora storitev, ki mora zagotavljati določeno kakovost (brezizgubnost paketov, pravilni vrstni red, čas prenosa itd.), zagotavljajo storitveni modeli teh omrežij različne modele zanesljivega prenosa podatkov. Ker za zanesljivost skrbi že sama arhitektura omrežja, so končni sistemi lahko preprosti (to je lahko npr. telefonski aparat, ki skrbi le še za zajem in predvajanje govora). Po drugi strani pa je dodajanje novih storitev za ta omrežja težavno, saj je zanje potrebna prilagoditev celotnega omrežja in ne le končnih naprav. Že iz tega opisa lahko razberemo, da Internet ne sodi v povezavna omrežja. Primeri povezavnih omrežnih protokolov so: ATM (angl. *Asynchronous Transfer Mode*), X.25, MPLS (angl. *Multiprotocol Label Switching*) in Frame Relay.

Izvedba komunikacije v povezavnih omrežjih ima tri faze, ki so **vzpostavitev povezave, prenos podatkov** in **rušenje povezave**. Na ta način se pred prenosom podatkov izvede vzpostavitev komunikacijskega kanala (voda) in s

tem zagotovljenega obsega omrežnih virov, ki so potrebni za kakovosten prenos. Vsak komunikacijski vod razprostira preko več zaporednih povezav. Na vsaki od teh povezav ima del voda svoj identifikator (**identifikator voda**), ki se lahko na različnih segmentih voda razlikuje. Administracija omrežja je na ta način preprosta, saj je možno posamezen usmerjevalnik konfigurirati neodvisno od drugih.

Usmerjevalniki izvajajo usmerjanje paketov na podlagi posredovalne tabele, ki vsebuje zapise v četverici (*vhodni vmesnik*, *vhodni ID voda*, *izhodni vmesnik*, *izhodni ID voda*). To posredovalno tabelo beremo na način: „*Če na podani vhodni vmesnik pride paket s podano vhodno številko voda, ga pošji na podani izhodni vmesnik s podano izhodno številko voda.*“ To pomeni, da so paketi označeni s številko vodov, na podlagi katere usmerjevalniki pakete ustreznost posreduje na izhodna vrata in spremenijo številko voda tako, da ustreza izhodni povezavi. Ponazoritev vsebine takšnih posredovalnih tabel si lahko ogledamo na primeru 4.2.

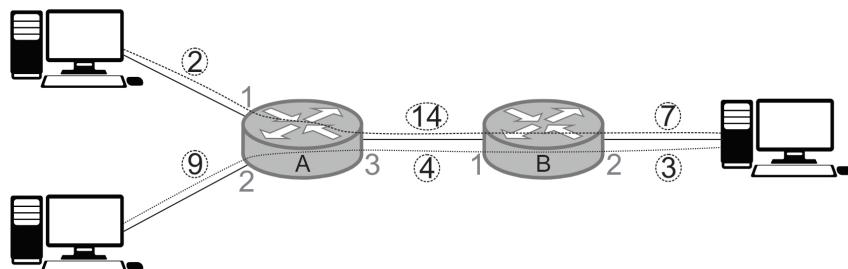
4.1.2 Nepovezavna omrežja

Pri nepovezavnih (ali *paketnih* oziroma *datagramskih*) omrežjih se prenos podatkov izvede brez predhodnega vzpostavljanja in kasnejšega rušenja povezave. Uporaba takšnih omrežij je primerna, kadar je ključnega pomena hitra izvedljivost komunikacije brez nujnih zagotovil o njeni kakovosti. V nepovezavnih omrežjih so zato pogoste *elastične storitve*, pri katerih je dopustna varianca v času prenosa. V primerjavi s povezavnimi omrežji je infrastruktura nepovezavnega omrežja preprostejša, zato je tudi dodajanje novih storitev in povezovanje heterogenih omrežij v njem preprosto. Breme storitev za zagotavljanje kakovosti prenosa pa se prenese na končne sisteme, ki morajo nadoknadi manjkajoča zagotovila kakovosti s strani omrežja.

Za razliko od povezavnih omrežij, kjer usmerjevalniki usmerjajo pakete na osnovi njihovih oznak, se v nepovezavnih omrežjih usmerjanje izvaja glede na ciljni naslov. Zato paketi vsebujejo naslov prejemnika, ki predstavlja ključni podatek, na podlagi katerega se vsak usmerjevalnik na poti odloči, na katera izhodna vrata posredovati dani paket. Kot bomo spoznali v razdelku 4.3, omrežni protokol Interneta (IP verzije 4) uporablja 32-bitne omrežne naslove. Ker je vseh možnih naslovov $2^{32} = 4,3 \cdot 10^9$, bi neučinkovita implementacija posredovalnih tabel morala hraniti zapise za vseh teh 2^{32} naslovov. Poleg težav s hranjenjem tako velikih tabel podatkov bi to upočasnjevalo posredovanje usmerjevalnikov, saj bi bilo pregledovanje tako velikega števila zapisov lahko časovno zelo zamudno. Zato bolj učinkovita implementacija posredovalnih tabel ne hrani podatka za vsak omrežni naslov, temveč za skupine omrežnih naslovov.

V poglavju o naslavljjanju v Internetu bomo spoznali tudi, da so naslovi IP hierarhični. To pomeni, da z upoštevanjem večjega števila bitov od začetka

★ **Primer 4.2:** Kot primer si poglejmo omrežje, ki je prikazano na spodnji sliki. Sive številke ob usmerjevalnikih označujejo številke njihovih vrat, obkrožene številke na povezavah pa identifikatorje vodov. V omrežju sta vzpostavljeni dve povezavi, in sicer med računalnikoma $R_1 \rightarrow R_3$ in med $R_2 \rightarrow R_3$.



Vzpostavitev teh povezav omogoča naslednja nastavitev posredovalnih tabel:

usmerjevalnik A

vhodni vmesnik	vhodni ID voda	izhodni vmesnik	izhodni ID voda
1	2	3	14
2	9	3	4

usmerjevalnik B

vhodni vmesnik	vhodni ID voda	izhodni vmesnik	izhodni ID voda
1	14	2	7
1	4	2	3

Na podlagi številke voda (3 ali 7) bo računalnik R_3 vedel, od katerega sogovornika (R_1 ali R_2) prihaja promet.

(levega konca) naslova proti desnemu vse bolj natančno opredeljujemo podomrežje Interneta, v katerem se vozlišče nahaja. Na primer, nekaj začetnih bitov lahko opredeljuje vrhovno organizacijo, ki razpolaga z naslovnim prostorom, naslednjih nekaj bitov lahko opredeljuje ponudnika omrežnih storitev, ki mu je prejšnja vrhovna organizacija dodelila del svojega prostora, in še nekaj nadaljnjih bitov lahko opredeljuje specifičnega odjemalca tega ponudnika storitev.

Tak način hierarhičnega usmerjanja napeljuje k temu, da lahko usmerjanje k skupini omrežnih naslovov (ki smo ga omenili v prejšnjem odstavku) implementiramo najbolj učinkovito kar na podlagi predpone omrežnega naslova, tako da pravila v posredovalni tabeli zapišemo kot par (*predpona ciljnega naslova, izhodna vrata*). V primeru, če se nek omrežni naslov ujema z več predponami (glej primer 4.3), pa se odločimo, da uporabimo **pravilo ujemanja najdaljše predpone**, ki zahteva, da paket posredujemo glede na najbolj specifično pravilo

★ **Primer 4.3:** Kot primer si poglejmo naslednjo posredovalno tabelo:

predpona ciljnega naslova	izhodni vmesnik
11001000 0001	1
11001100 111	2
11001100 111110	3
vsi ostali	4

Ta posredovalna tabela določa, da bo usmerjevalnik, ki prejme paket z naslovom 11001000 00010000 10101111 11101110 le-tega posredoval na izhodna vrata 1, paket za naslov 11001000 11111010 10111001 10011010 pa na izhodna vrata 3. Vidimo, da čeprav se predpona zadnjega naslova ujema tudi z 2. pravilom v posredovalni tabeli, se na podlagi pravila ujemanja najdaljše prepone upošteva 3. pravilo.

v omrežni hierarhiji.

Poleg posredovanja morajo usmerjevalniki opravljati tudi nalogu usmerjanja. To izvajajo z **usmerjevalnimi algoritmi in protokoli**, ki skrbijo za posodabljanje posredovalnih tabel glede na stanje omrežja. Pri nepovezavnih omrežjih je delo usmerjevalnih algoritmov iterativno (usmerjevalniki si izmenjujejo sporočila na vsakih nekaj sekund) in počasno v primerjavi z vzpostavitenim časom vodov pri povezavnih omrežjih (ta traja nekaj mikrosekund).

4.2 Storitve omrežne plasti

Podobno, kot smo našeli storitve povezavne plasti, naštejmo tudi storitve, ki jih lahko zagotavlja omrežna plast:

1. **zagotovljena dostava paketov:** omrežna plast lahko s sistemom potrjevanja in zahtev po ponovnem pošiljanju skrbi za zagotavljanje pravilnega sprejetja paketov na strani prejemnika;
2. **dostava v pravilnem zaporedju:** ker lahko paketi skozi paketno omrežje potujejo po različnih poteh, lahko prispejo v drugačnem zaporedju, kot so bili poslani. Omrežna plast lahko zagotavlja mehanizem za urejanje paketov v pravilni vrstni red;
3. **dostava v zagotovljenem času:** skrb za to, da čas potovanja paketa skozi omrežje ne preseže zgornje časovne meje. To je pomembno v primerih, kadar aplikacija potrebuje prenos v stvarnem (realnem) času, kot je npr. telefonija oziroma prenos govora;
4. **zagotovljena spodnja meja pasovne širine:** skrb za najmanjšo hitrost prenosa skozi omrežje. Podobno kot pri prejšnji storitvi so takšna zagotovila

pomembna za aplikacije, ki morajo ustvariti občutek stvarnosti. Takšne aplikacije so prenos zvoka in slike, lahko pa tudi omrežne igre, ki zahtevajo intenzivno komunikacijo;

5. **omejitev variance zakasnitve med zaporednimi paketi** (angl. *jitter*): zagotavlja pošiljanje podatkov v stalnem toku. Če dva zaporedna paketa označimo s p_1 in p_2 , čas pošiljanja in prejetja pa s $t_{pošiljanja}$ in $t_{prejetja}$, lahko to zagotovilo zapišemo kot

$$t_{pošiljanja}(p_2) - t_{pošiljanja}(p_1) \approx t_{prejetja}(p_2) - t_{prejetja}(p_1);$$

6. **varna komunikacija**: različne storitve za zagotavljanje zaupnosti, avtentifikacije, avtorizacije, integritete podatkov in drugih oblik omrežne varnosti, o katerim bomo govorili v poglavju 7.

Podobno kot lahko različni povezavni protokoli zagotavljajo različne nabore storitev, je enako tudi s protokoli na omrežni plasti. Naboru storitev, ki jih implementira nek protokol, pravimo **storitveni model**. Kot zanimivost primerjajmo storitvene modele protokola IP (glavni omrežni protokol v Internetu) in protokola ATM (angl. *Asynchronous Transfer Mode*, ki je od zgodnjih 1980. let temeljni omrežni protokol javnega telefonskega omrežja. Pakete protokola ATM imenujemo tudi *celice*, ki so za razliko od paketov IP vse enako velike. Medtem, ko je protokol IP *nepovezaven* protokol, je protokol ATM *povezaven*. Pri uporabi protokola ATM lahko izberemo enega od štirih načinov izvajanj protokola, ki zagotavlja kakovost prenosa na drugačen način: CBR (konstantna bitna hitrost, angl. *constant bit rate*), ABR (razpoložljiva bitna hitrost, angl. *available bit rate*), VBR (spremenljiva bitna hitrost *variable bit rate*), UBR (neopredeljena bitna hitrost, angl. *unspecified bit rate*). Tabela 4.1 prikazuje primerjavo storitvenih modelov protokola IP in ATM v načinu izvajanja CBR in ABR.

STORITEV	IP	ATM (CBR)	ATM (ABR)
zagotovljena pasovna širina	ne	da, konstantna	da, minimalna
preverba izgube paketov	ne	da	ne
kontrola vrstnega reda paketov	ne	da	da
zagotovljen čas dostave	ne	da	ne
kontrola zamašitve	ne	ni potrebe	da

Tabela 4.1 Primerjava storitvenih modelov protokolov IP in ATM

Iz primerjave storitvenih modelov vidimo, da protokol IP ne zagotavlja nobene od naštetih storitev. Za tak storitveni model pravimo, da je model zagotavljanja storitev **po najboljših zmožnostih** (angl. *best-effort service*). To pomeni, da

protokol IP zagotavlja le osnovne storitve prenosa paketov na omrežni plasti, za njegove manjkajoče storitve pa bo moral poskrbeti nek drug protokol z višjih plasti komunikacijskega modela. V poglavju 5 bomo preučili, da na transportni plasti to opravi protokol TCP.

Protokol IP je v Internetu edini omrežni protokol. Ker uporablja storitveni model zagotavljanja storitev po najboljših zmožnostih, mu velikokrat pravimo tudi, da predstavlja *ozko grlo Interneta*, kot smo to prikazali v uvodnem poglavju na sliki 1.8.

4.3 Protokol IPv4

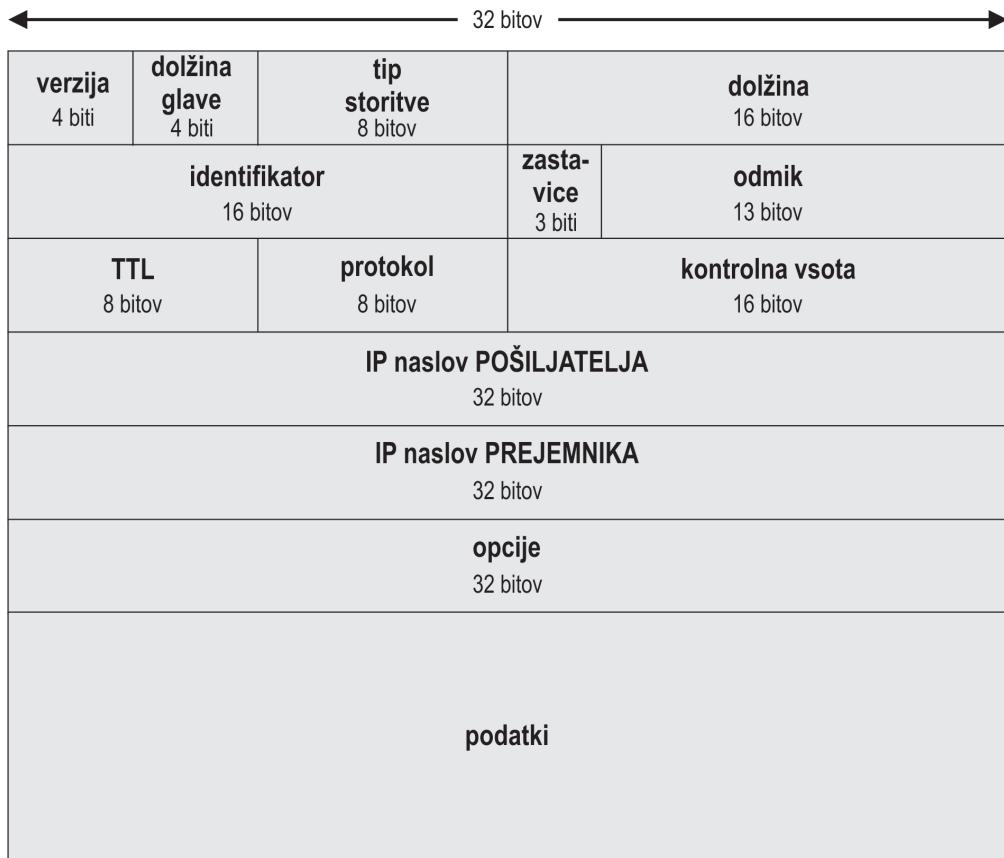
Protokol IP (angl. *Internet Protocol*) je v Internetu zagotovo najpomembnejši protokol na omrežni plasti. Je protokol, ki zagotavlja prenos enkapsuliranih transportnih segmentov v *pakete* preko omrežne infrastrukture. Trenutno najbolj razširjena verzija tega protokola je verzija 4 (*IPv4*), ki je bila zasnovana leta 1981. Paket protokola IPv4 je prikazan na sliki 4.1, ta pa vsebuje polja, ki so razložena v tabeli 4.2.

4.3.1 Fragmentacija

Fragmentacija je postopek, ki omogoča delitev velikega paketa IP na manjše pakete – te imenujemo **fragmenti**. Potreba za delitev paketov na manjše izhaja iz omejitve dolžine okvirjev na povezavni plasti. Vemo, na primer, da je privzeta največja velikost okvirja (parameter MTU) pri protokolu Ethernet enaka 1500 Byte-ov, pri protokolu 802.11 pa 7981 Byte-ov. Ker lahko paket med potovanjem po omrežju prečka več različnih povezav, se lahko MTU med potovanjem tudi spreminja. Zato je fragmentacija postopek, ki ga lahko izvede tudi vsak usmerjevalnik na poti, da prilagodi velikost paketov prenosa po naslednji povezavi. Ko vsi fragmenti pripotujejo do prejemnika, njegova omrežna plast izvede združevanje fragmentov v celotni paket, nato pa ga preda transportni plasti.

V paketu IP se za izvedbo fragmentacije uporablajo naslednja polja:

- **ID**: identifikator fragmentov; fragmenti, ki so nastali z delitvijo istega paketa, imajo isto vrednost identifikatorja;
- **ZASTAVICE**: 3 biti, od katerih sta zadnja dva pomembna za izvedbo fragmentacije. Prvi nosi oznako **DF** (angl. *don't fragment*) in preprečuje izvedbo fragmentacije, kar je uporabno za odpravljanje težav, če prejemnik ne zna fragmentov rekonstruirati nazaj v celoto, ali pa za iskanje najmanjšega MTU na poti do prejemnika. Drugi bit nosi oznako **MF** (angl. *more fragments*) in opredeljuje, da trenutnemu fragmentu sledijo še nadaljnji fragmenti;



Slika 4.1 Paket protokola IPv4

- **ODMIK:** 13-bitno polje, ki opredeljuje, kateri del podatkov iz prvotnega paketa hrani trenutni fragment. Odmik je podan v vrednosti, ki je večkratnik enote 8 bajtov, odmik prvega fragmenta (začetek podatkov) pa ima vrednost 0. Ker je odmik podan v enotah 8B, je tudi dolžina podatkov v vsakem fragmentu deljiva z 8. V primeru, da količina podatkov ni deljiva z 8, moramo odmik zaokrožiti navzdol tako, da dolžina celega fragmenta ne presega vrednosti MTU.

Način uporabe polj za fragmentacijo si lahko ogledamo na primerih 4.4 in 4.5.

~~~~~ **Varnostni pomislek:** Če postopka za določanje odmika podatkov ne izvedemo pravilno, lahko pri rekonstrukciji fragmentov v celoten paket dosežemo nepredvidljive učinke. To dejstvo lahko izkoriščajo tudi omrežni napadalci, ki z izvebo **napadov s fragmentacijo** (angl. *fragmentation attack*) poskušajo *one-mogočiti delovanje ciljnih sistemov* (angl. *Denial of Service - DoS*). Dva takšna napada sta:

★ **Primer 4.4:** Denimo, da imamo paket velikosti 3500 B, ki ga želimo poslati po povezavi, na kateri velja MTU = 1500 B. Vemo, da paket, velik 3500 B, sestavljajo glava v velikosti 20 B (predpostavimo, da se opcije ne uporabljajo) in podatki v velikosti 3480 B. Slednje podatke bomo morali razdeliti na fragmente, katerih skupna velikost (glava + podatkovna vsebina) ne bo presegla 1500 B. Označimo izvorni paket s simboličnim identifikatorjem "x", na podlagi česar lahko tudi fragmente označeni z istim identifikatorjem. Podatke po fragmentih razporedimo na naslednji način:

- V prvi fragment razporedimo prvih 1480 B podatkov, kar skupaj z glavo (20 B) znaša 1500 B. V polja za fragmentacijo prvega fragmenta zapišemo:

|                 |      |         |
|-----------------|------|---------|
| identifikator=x | MF=1 | odmik=0 |
|-----------------|------|---------|

Ker smo v prvi fragment razporedili 1480 B, jih moramo v preostale razporediti še  $3480B - 1480B = 2000B$ .

- V drugi fragment razporedimo nadaljnjih 1480 B podatkov, za tretji jih ostane še 520 B. Teh 1480 B podatkov mora nadaljevati podatke iz prejšnjega fragmenta, zato opredelimo odmik (v enotah 8 bajtov) enak  $1480B/8B = 185$ . V polja drugega fragmenta zapišemo:

|                 |      |           |
|-----------------|------|-----------|
| identifikator=x | MF=1 | odmik=185 |
|-----------------|------|-----------|

- V tretji fragment razporedimo še preostanek podatkov, označimo, da gre za zadnji fragment, in polja fragmenta zapišemo kot:

|                 |      |           |
|-----------------|------|-----------|
| identifikator=x | MF=0 | odmik=370 |
|-----------------|------|-----------|

★ **Primer 4.5:** Denimo, da imamo paket velikosti 3500 B, ki ga želimo poslati po povezavi, na kateri velja MTU = 2200 B. Paket, velik 3500 B hrani 3480 B podatkov (opcije se ne uporabljajo). Podatke po fragmentih razporedimo na naslednji način:

- V prvi fragment bi želeli razporediti prvih  $2200B - 20B = 2180B$  podatkov. To bi pomenilo, da bi drugi fragment želeli nadaljevati z neveljavnim odmikom  $2180B/8B = 272,75$ , ki ni celo število. Ker takšnega odmika ne moremo uporabiti, lahko v prvi fragment shranimo manj podatkov, t.j.  $272 * 8B = 2176B$ . V polja za fragmentacijo prvega fragmenta zapišemo:

|                 |      |         |
|-----------------|------|---------|
| identifikator=x | MF=1 | odmik=0 |
|-----------------|------|---------|

- Ker smo v prvi fragment razporedili  $2176B$ , moramo v drugi fragment razporediti še  $3480B - 2176B = 1304B$ . Fragment tvorimo tako, da v polja zapišemo:

|                 |      |           |
|-----------------|------|-----------|
| identifikator=x | MF=1 | odmik=272 |
|-----------------|------|-----------|

| POLJE                   | DOLŽINA              | POMEN                                                                                                                                                                                                                                                                                                                                                   |
|-------------------------|----------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| VERZIJA                 | 4 biti               | Verzija protokola IP (ima vrednost 4).                                                                                                                                                                                                                                                                                                                  |
| DOLŽINA GLAVE           | 4 biti               | Določa velikost glave v 32-bitnih besedah in s tem podaja, kje se začnejo podatki (enkapsulirana vsebina paketa).                                                                                                                                                                                                                                       |
| TIP STORITVE            | 8 bitov              | Oznaka vrste paketa, ki omogoča razlikovanje storitev in zagotavljanje kakovosti storitev (npr. za prenos v stvarnem času)                                                                                                                                                                                                                              |
| DOLŽINA                 | 16 bitov             | Skupna dolžina celega paketa v bajtih, običajno vrednost 1500 bajtov.                                                                                                                                                                                                                                                                                   |
| ID, ZASTAVICE, ODMIK    | 32 bitov             | Polja, ki se uporabljajo za izvedbo fragmentacije (delitev večjih paketov na manjše, podrobneje v razdelku 4.3.1).                                                                                                                                                                                                                                      |
| TTL                     | 8 bitov              | Življenski čas paketa, merjen v številu skokov (omrežnih povezav na poti). Vsak usmerjevalnik na poti zmanjša vrednost TTL za 1, kadar pa TTL doseže vrednost 0, usmerjevalnik paket zavrije. Na ta način preprečimo morebitno neskončno ciklanje paketa skozi omrežje.                                                                                 |
| PROTOKOL                | 8 bitov              | Oznaka protokola, ki je enkapsulirana v paketu. Evidenca možnih vrednosti je dostopna na <a href="http://www.iana.org/assignments/protocol-numbers/protocol-numbers.xhtml">http://www.iana.org/assignments/protocol-numbers/protocol-numbers.xhtml</a>                                                                                                  |
| KONTROLNA VSOTA         | 16 bitov             | Kontrolna vsota, izračunana z algoritmom <i>Internet Checksum</i> , ki pokriva samo glavo paketa. Zaradi spremenjanja polja TTL jo mora na novo izračunati vsak usmerjevalnik na poti.                                                                                                                                                                  |
| IP NASLOV IZVORA/PONORA | 32 + 32 bitov        | Omrežni naslov pošiljatelja in prejemnika.                                                                                                                                                                                                                                                                                                              |
| OPCIJE                  | 32 bitov             | Možne nestandardne uporabniške razširitve glave paketa z dodatnimi polji. Opcije se uporabljajo le redko, ker njihova uporaba poveča čas procesiranja paketa in premakne začetek podatkov na kasnejše mesto v paketu, ki ga je potrebno poiskati z branjem polja DOLŽINE GLAVE. Če opcij ne uporabljamo, ima glava paketa IP dolžino natanko 20 bajtov. |
| PODATKI                 | spremenljiva dolžina | Enkapsulirani podatki transportne plasti, v Internetu je to običajno segment protokola TCP ali datagram protokola UDP.                                                                                                                                                                                                                                  |

Tabela 4.2 Pomen polj v paketu protokola IPv4

- **Napad s prekrivajočimi se fragmenti** (angl. *overlapping fragment attack*): Napadalec določi polja odmikov tako, da podatki kasnejših fragmentov prepišejo podatke prejšnjih (vrednosti odmikov so torej prenizke). Če prejemnik ni pripravljen, da nepravilno oblikovane fragmente zavrije, lahko to povzroči napako v njegovegem komunikacijskem skladu in prekinitev delovanja.
- **Napad z drobnimi fragmenti** (angl. *tiny fragment attack*): Napadalec razkosa prvotni paket na tako majhne fragmente, ki so manjši od glave (20 B). S tem lahko bodisi zmede prejemnika, ki pričakuje pakete, dolge vsaj 20 B, ali pa omogoči prenos podatkov, pri katerih varnostne naprave (npr. požarni zidovi) ne morejo pravilno zaznati vsebine (npr. naslov pošiljatelja),

**★ Primer 4.6:**

|                                                       |                                     |     |     |   |
|-------------------------------------------------------|-------------------------------------|-----|-----|---|
| primer naslova IP (dvojiško):                         | 110111110010001111000100000010      |     |     |   |
| primer naslova IP (dvojiško), razdeljenega na oktete: | 11011111 10010001 11110001 00000010 |     |     |   |
| okteti, pretvorjeni v desetiške vrednosti:            | 223                                 | 145 | 241 | 2 |
| zapis naslova IP                                      | 223.145.241.2                       |     |     |   |

na podlagi katere bi paket lahko zavrgli.

Izvedbo obeh napadov lahko preprečimo s posodobitvijo komunikacijskega sklada prejemnika tako, da upošteva te izredne oblike fragmentov in jih zavrže.

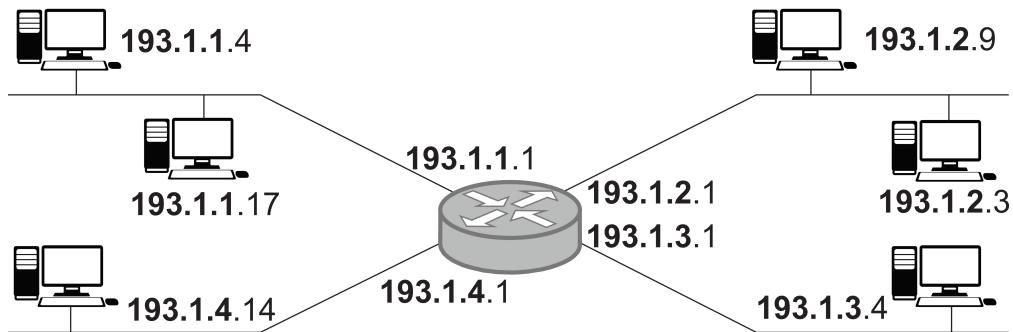
### 4.3.2 Naslavljanje IPv4

V poglavju 4.1.2 smo že omenili, da uporablja protokol IP (ki je *paketni, nepovezavni* ali *datagramski*) 32-bitne, hierarhične naslove. Za razliko od fizičnih (MAC) naslovov, ki morajo biti enolični le na določenem mediju, morajo biti omrežni naslovi IP *globalno enolični* v celotnem omrežju. Naslov IP, ki je sestavljen iz 32 bitov, običajno razdelimo na štiri osemitbitne bloke, imenovane *oktete*, nato pa vsakega pretvorimo v desetiško vrednost. Štiri oktete ločimo s pikami, kot je prikazano na primeru 4.6.

#### Delitev na podomrežja

Unikatnost in dolžina naslovov IP nam omogočata nedvoumno naslavljanje približno 4 milijard ( $2^{32}$ ) naslovnikov, hierarhija naslovov pa nam zagotavlja izvedbo učinkovitega usmerjanja na podlagi predpon naslovov. V svetu naslove organiziramo tako, da jih ločimo v različna *podomrežja* in vsakemu podomrežju določimo svojo predpono naslova. Iz tega sledi, da naslov IP delimo na dva dela, kot to prikazuje slika 4.2. Vsi računalniki znotraj podomrežja imajo isto predpono (ozioroma naslov podomrežja) in različne pripone naslova (ozioroma naslove vmesnikov v podomrežju). Ta spominja na analogijo iz vsakodnevnega življenja: v mestih so hiše organizirane v ulice tako, da imajo imajo vse hiše isti naslov ulice (naslov podomrežja), le drugo hišno številko (naslov vmesnika). Slika 4.2 prikazuje primer takšne delitve omrežja v podomrežje.

Delitev v podomrežja doprinaša dve prednosti. Prvič, velja, da so vsi računalniki znotraj nekega podomrežja medseboj lokalno dosegljivi brez posredovanja usmerjevalnika (med njimi torej obstaja povezljivost na povezavni plasti, npr. z omrežnim stikalom). Drugič, pri delitvi naslovnega prostora v podomrežja lahko več podomrežij hierarhično združujemo v večja omrežja. Ta druga lastnost pa



**Slika 4.2** Z usmerjevalnikom lahko omrežje razdelimo na manjša podomrežja. V vsakem podomrežju imajo naslovi isto predpono (označena odebeleno), številke vmesnikov v tem podomrežju pa so si različne. Če je podomrežje segment lokalnega omrežja, so računalniki v njem lahko povezani z omrežnim stikalom in medseboj dostopni. S slike lahko opazimo, da imajo tudi vrata usmerjevalnika svoje omrežne naslove, saj delujejo na omrežni plasti.

nam omogoča učinkovito implementirati usmerjevanje, kot bomo to spoznali v poglavju 4.3.3.

### Naslavljanje z uporabo razredov naslovov

Tudi postopek naslavljanja v Internetu je skozi čas doživel svoj napredek, dokler ni leta 1993 prišel do naslavljanja po principu CIDR (angl. *Classless Inter-Domain Routing*). Predhodno, od leta 1981 se je v Internetu uporabljalo naslavljanje na osnovi t.i. **razredov naslovov**, ki so bili definirani, kot prikazuje tabela 4.3.

| razred | začetni biti naslova | število bitov za naslov podomrežja (število možnih podomrežij) | število bitov za naslov vmesnika (število možnih vmesnikov) |
|--------|----------------------|----------------------------------------------------------------|-------------------------------------------------------------|
| A      | 0                    | 7 ( $128 = 2^7$ )                                              | 24 ( $16.777.216 = 2^{24}$ )                                |
| B      | 10                   | 16 ( $16.384 = 2^{14}$ )                                       | 16 ( $65.536 = 2^{16}$ )                                    |
| C      | 110                  | 24 ( $2.097.152 = 2^{21}$ )                                    | 8 ( $256 = 2^8$ )                                           |

**Tabela 4.3** Razredi naslovov za neposredno (unicast) naslavljanje pred pričetkom naslavljanja po principu CIDR. (Opomba: Obstajata tudi razreda D in E, ki pa se uporablja za razpošiljanje (multicast) in možne razširitve)

Iz definicij razredov vidimo, da je v naslovni prostor nekega podomrežja bilo možno priključiti bodisi 16 milijonov različnih vmesnikov (razred A), 65 tisoč različnih vmesnikov (razred B) ali pa le 256 (razred C). Pri dodeljevanju IP naslovnih prostorov ponudnikom Interneta, podjetjem, univerzam, sta se razreda A in B zato pogosto izkazala kot prevelika, saj redkokateri odjemalec potrebuje naslovni prostor za podomrežje, v katerega bo priključil do 65 tisoč naprav.

Po drugi strani pa se je kapaciteta razreda C (256 naprav) pogosto izkazala za premajhno, saj so jo potrebe srednje velikih in velikih podjetij, univerz in raziskovalnih organizacij hitro presegle. Posledično je odjemalcem bilo pogosteje potrebno dodeliti naslovni prostor razreda B, pri katerem pa je veliko število naslovov ostajalo neizkoriščeno, če je le odjemalec imel manj kot 65 tisoč naprav. Takšno razdeljevanje naslovnega prostorov je bilo potratno in je že v zgodnjih 90. letih kazalo na možnosti pomanjkanja.

### Naslavljanje po principu CIDR

Za bolj smotorno ravnanje s prostim naslovnim prostorom so se snovalci Interneta odločili zmanjšati razlike med kapacitetami naslovnih prostorov razredov A, B in C ter s tem v številu možnih podomrežij. To so naredili z uvedbo naprednejšega načina naslavljanja CIDR (angl. *Classless Inter-Domain Routing*), ki omogoča, da lahko rezerviramo poljubno dolgo predpono naslova za zapis naslova podomrežja (in ne nujno samo 8, 16 ali 24 bitov). Črka C v kratici CIDR poudarja torej izboljšavo naslavljanja (*Classless*), ki ne uporablja več omejenih razredov. Zato, da lahko opredelimo, kateri del naslova IP določa naslov podomrežja in kateri naslov vmesnika, potrebujemo še dodaten podatek, ki ga imenujemo **omrežna maska**. Omrežna maska je ravno tako 32-bitno število, ki ima enice na tistih mestih, ki v pripadajočem naslovu IP predstavljajo naslov podomrežja in ničle na tistih mestih, ki predstavljajo naslov vmesnika. Z uporabo omrežne maske lahko iz naslova IP ugotovimo naslov omrežja tako, da na vse bite naslova IP, kjer ima omrežna maska soležni bit enak 0, postavimo na 0.

Znotraj posameznega podomrežja lahko s preostalimi biti (tistimi, ki imajo na soležnem mestu v omrežni maski vrednost 0) naslovimo različne omrežne vmesnike. Od vseh naslovov sta dva rezervirana za posebne namene: naslov iz samih ničel, ki običajno predstavlja naslov omrežja in ne posameznega vmesnika, in naslov iz samih enic, ki se uporablja za naslavljanje vseh vmesnikov znotraj podomrežja (t.i. *broadcast* naslov, podobno kot na povezavni plasti).

#### 4.3.3 Hierarhija pri naslavljjanju

V Internetu je neprofitna organizacija ICANN (angl. *Internet Corporation for Assigned Names and Numbers*) tista, ki ponudnikom Interneta (angl. *Internet Service Provider, ISP*) dodeljuje naslovne prostore. To naredi tako, da posameznemu ponudniku dodeli podomrežje z določenim naslovom. Ker vsak ponudnik svobodno razpolaga z dodeljenim naslovnim prostorom, se lahko odloči dodeljeno podomrežje razdeliti na manjša podomrežja. To storiti tako, da se znotraj svojega naslovnega prostora odloči dodatno nameniti nekaj bitov označevanju podomrežij, ki so v njegovi domeni, preostale bite pa prepusti številčenju vmesnikov.

★ **Primer 4.7:** Kot primer si poglejmo, kako zapisati naslov podomrežja, vmesnika in omrežno masko za naslov IP 223.145.251.112, v katerem prvih 20 bitov določa naslov omrežja (pike pri binarnem zapisu so zapisane zaradi lažje berljivosti):

|                           |                                                         |
|---------------------------|---------------------------------------------------------|
| naslov IP:                | 223.145.251.112<br>11011111.10010001.1111011.01110000   |
| omrežna maska:            | 11111111.11111111.11110000.00000000<br>/20              |
| naslov omrežja:           | 11011111.10010001.11110000.00000000<br>223.145.240.0/20 |
| zapis naslova z masko:    | 223.145.251.112/20                                      |
| broadcast naslov:         | 11011111.10010001.11111111.11111111<br>223.145.255.255  |
| najnižji naslov vmesnika: | 11011111.10010001.11110000.00000001<br>223.145.240.1    |
| najvišji naslov vmesnika: | 11011111.10010001.11111111.11111110<br>223.145.255.254  |
| možno število vmesnikov:  | $2^{12} - 2 = 4094$                                     |

★ **Primer 4.8:** Kot primer si poglejmo ponudnika Interneta, ki mu je ICANN dodelil naslovni prostor 200.23.16.0/20. Denimo, da želi ponudnik svoj naslovni prostor naprej deliti največ 8 podjetjem, za kar bo potreboval tri dodatne bite ( $8 = 2^3$ ). Dodatni biti, na podlagi katerih lahko razločimo naslove omrežij različnih podjetij, so podčrtani dvojno. Dodeljene naslove lahko zapišemo kot:

|                                 |                                                       |
|---------------------------------|-------------------------------------------------------|
| naslovni prostor ponudnika:     | 200.23.16.0/20<br>11001000.00010111.00010000.00000000 |
| naslovni prostor za Podjetje 1: | 11001000.00010111.00010000.00000000<br>200.23.16.0/23 |
| naslovni prostor za Podjetje 2: | 11001000.00010111.00010010.00000000<br>200.23.18.0/23 |
| ...                             | ...                                                   |
| naslovni prostor za Podjetje 8: | 11001000.00010111.00011110.00000000<br>200.23.30.0/23 |

Usmerjevalniki, ki v Internetu izvajaju usmerjanje na podlagi predpon naslovov, lahko takšno hierarhično organizacijo naslovov s pridom izkoristijo za enostavnejši zapis posredovalnih tabel. Usmerjevalnikom v Internetu namreč ni potrebno vedeti ničesar o posameznih naslovnih prostorih podomrežij, ki jih je ponudnik Interneta po svojih željah delil na manjše kose, ampak mu lahko posredujejo ves promet, ki se začne s predpono njegovega celotnega naslovnega prostora. Zadošča torej, da je šele usmerjevalnik ponudnika Interneta nastavljen tako, da zna prejeti promet posredovati posameznemu podomrežju. Na ta način ostaneta tako določanje podomrežij kot tudi usmerjanje vanje v celoti v administrativni domeni ponudnika Interneta.

#### 4.3.4 Prevajanje omrežnih naslovov

Zaradi naraščanja števila priključenih naprav se je izkazalo, da bo 32-bitni naslovni prostor, ki ga zagotavlja protokol IPv4, kmalu porabljen. Čeprav je k počasnejši porabi naslovnega prostora prispevala bolj smotrna poraba z uvedbo naslavljanja CIDR, je septembra 2012 članska organizacija ICANN, imenovana RIPE NCC<sup>1</sup>, ki je odgovorna za dodeljevanje naslovnega prostora na področju Evrope, držav bližnjega vzhoda in centralne Azije, tudi dejansko dodelila še zadnje svoje razpoložljive bloke naslovnega prostora.

Da bi snovalci Interneta upočasnili hitrost porabe javnega naslovnega prostora, so v zgodnjih 90. letih predlagali uvedbo *lokalnih (zasebnih) naslovnih prostorov in prevajanja omrežnih naslovov* (angl. *Network Address Translation, NAT*), kjer se več zasebnih naslovov lahko prevede v enega javnega. *Lokalni* ali *zasebni naslovi* so naslovi iz vnaprej definiranih skupin, prikazanih v tabeli 4.4. Za te naslove velja, da:

- ne nastopajo v javnem Internetu, ampak so namenjeni uporabi v lokalnih omrežjih,
- so za razliko od javnih naslovov IP lahko ponovljivi v različnih lokalnih omrežjih (in morajo torej biti le unikatni lokalno, ne pa tudi globalno),
- če želimo vmesnikom z lokalnimi naslovi omogočiti dostop do javnega Interneta, moramo zagotoviti preslikavo lokalnega naslova v javni.

| zasebni naslovi               | omrežje/maska  | število naslovov |
|-------------------------------|----------------|------------------|
| 10.0.0.0 - 10.255.255.255     | 10.0.0.0/8     | $2^{24}$         |
| 172.16.0.0 - 172.31.255.255   | 172.16.0.0/12  | $2^{20}$         |
| 192.168.0.0 - 192.168.255.255 | 192.168.0.0/16 | $2^{16}$         |

**Tabela 4.4** Zasebni naslovi IP

<sup>1</sup>Réseaux IP Européens Network Coordination Centre

Tretjo od zgornjih alinej (preslikovanje zasebnih naslovov v javne) izvaja funkcija preslikovanja naslovov (NAT), ki jo opravljajo usmerjevalniki, saj so to tiste naprave, ki povezujejo lokalna omrežja z javnimi. Usmerjevalnik hrani preslikave v *preslikovalni tabeli NAT*, ki hrani zapise oblike:

|                             |                               |
|-----------------------------|-------------------------------|
| javni naslov, številka vrat | zasebni naslov, številka vrat |
|-----------------------------|-------------------------------|

Novi podatek, ki se hrani v tabeli NAT in ki ga dosedaj še nismo omenili, je *številka vrat*. To je podatek, ki ga bomo natančno spoznali v poglavju 5, ki govorji o transportni plasti. Številka vrat se uporablja za naslavljanje aplikacijskega procesa znotraj računalnika (torej je neke vrste "podnaslov" aplikacije znotraj računalnika, ki pa ga že znamo nasloviti z naslovom IP). Pri preslikovanju naslovov je ta dodatni podatek potreben zato, ker mora usmerjevalnik več notranjih zasebnih naslovov preslikati v enega javnega (na ta način varčujemo z javnim naslovnim prostorom) in zato, da lahko loči, kateremu zasebnemu naslovu je namenjen določen paket, uporablja za "podnaslavljanje" naprav poleg naslovov IP še različne številke vrat. Način delovanja mehanizma NAT si lahko ogledamo na primeru 4.9.

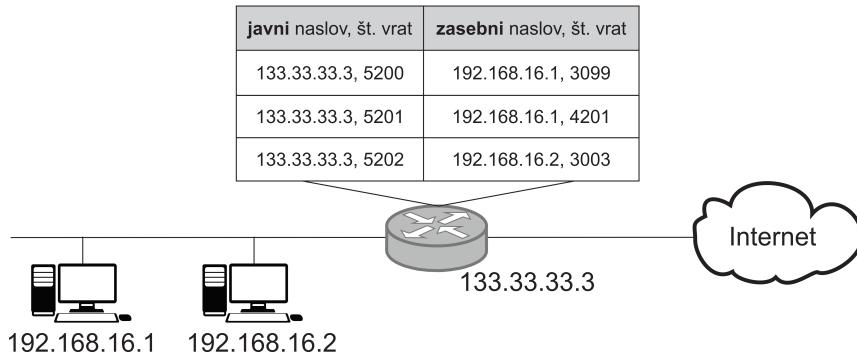
NAT ima svoje prednosti in slabosti. Njegove dobre lastnosti so, da očitno zmanjšuje porabo javnih naslovov, saj je možno več zasebnih naslovov (do 65536, kolikor je možnih številk vrat) preslikovati v en sam javni naslov. Ker onemogoča dostopanje z Interneta do lokalnih računalnikov, ki niso prvi poslali zahtevka (odhodni promet), ta mehanizem povečuje tudi varnost računalnikov v zasebnih omrežjih. Prednost zasebnega naslovnega prostora je tudi ta, da lahko lokalni administratorji spreminjajo lokalne naslove neodvisno od zunanjih javnih naslovov.

Kljud temu pa je potrebno poudariti, da je koncept NAT zasnovan v nasprotju z načeli snovanja Interneta, saj uporablja številko vrat kot del podatka za naslavljanje lokalnih računalnikov in ne za naslavljanje aplikacijskih procesov, kot je njen namen. Zato pravimo, da NAT krši t. i. *princip končnih sistemov* (angl. *end-to-end principle*), ki zahteva njihovo preprosto medsebojno naslavljanje z uporabo naslovov IP. Preslikovanje naslovov ravno tako dodatno obremenjuje usmerjevalnike, ki morajo pri protokolih iz družine P2P (angl. *Peer-to-Peer*) hrani veliko število preslikav. Problem pri protokolih iz družine P2P se pojavlja tudi, kadar želi veliko število računalnikov z Interneta prvih dostopiti do računalnika v notranjem omrežju, kar pa ni mogoče pred prvotnim pošiljanjem zahtevka navzven in dodajanjem zapisa v tabelo NAT. Kot boljšo rešitev za odpravo težav s pomanjkanjem naslovnega prostora je zato smiselno uporabiti protokol IPv6.

## 4.4 Protokol IPv6

Z rastjo Interneta se je izkazalo, da je težav s protokolom IP verzije 4, ki je bil razvit leta 1981, več. Med te težave sodijo:

★ **Primer 4.9:** Kot primer si poglejmo preslikovanje naslovov NAT, ki ga izvaja usmerjevalnik, ki ločuje lokalno omrežje dveh računalnikov (zasebni naslovni prostor) in Internet, kot prikazuje naslednja slika:



Primer možne komunikacije, pri kateri opazujmo spremembe v tabeli NAT, je:

1. Na začetku je tabela NAT prazna. Možna je le komunikacija navzven (iz lokalnega omrežja v Internet), saj zaradi prazne tabele NAT usmerjevalnik ne ve, kateremu lokalnemu računalniku je prispeli paket namenjen.
2. Denimo, da računalnik z naslovom 192.168.16.1 pošlje paket v Internet. Kot bomo spoznali v poglavju o transportni plasti, je vsak poslani podatek opremljen s *številko izvornih vrat*, ki pove, katera aplikacija pričakuje odgovor na poslani zahtevek. Ob prejemu zahtevka bo usmerjevalnik dodal v tabelo zapis:

| javni naslov, številka vrat | zasebni naslov, številka vrat |
|-----------------------------|-------------------------------|
| 133.33.33.3, 5200           | 192.168.16.1, 3099            |

in v paketu, ki ga je poslal računalnik 192.168.16.1 z vrat 3099, spremenil naslov pošiljatelja v 133.33.33.3, 5200 ter ga posredoval strežniku.

3. Ko strežnik obdela zahtevek, pošlje svoj odgovor in ga naslovi prejemniku, ki mu ga je poslal - torej na naslov 133.33.33.3, 5200. Ko usmerjevalnik prejme njegov paket, bo lahko z vpogledom v tabelo NAT razbral, da je ta paket pravzaprav namenjen naslovu 192.168.16.1, 3099; zato bo v paketu zamenjal naslov prejemnika in prejemniku posredoval paket. Ko bo računalnik 192.168.16.1 paket prejel, bo vso opisano dogajanje zanj transparentno, saj je preslikovanje in zamenjavo naslovov opravil usmerjevalnik brez njegove vednosti.
4. Denimo, da ob kasnejšem času računalnik 192.168.16.1 želi dostopiti še do nekega drugega strežnika. V tabelo NAT se bo za ta drugi dostop dodal nov zapis, ki se bo razlikoval v številki izvornih in preslikanih vrat. Na ta način bo računalnik 192.168.16.1 lahko prejema dohodni promet od dveh strežnikov, pri čemer bosta tako usmerjevalnik kot tudi prejemnik lahko odgovore razlikovala glede na označeno številko vrat.
5. Podobna komunikacija z dopolnitvijo tabele NAT se lahko odvije tudi s strani računalnika 192.168.16.2. Vendar pa pred tem ta računalnik ne more prejeti nobenega dohodnega prometa z Internetom, saj v usmerjevalniku ni zapisa, ki omogoča preslikavo ciljnega naslova v naslov računalnika 192.168.16.2.

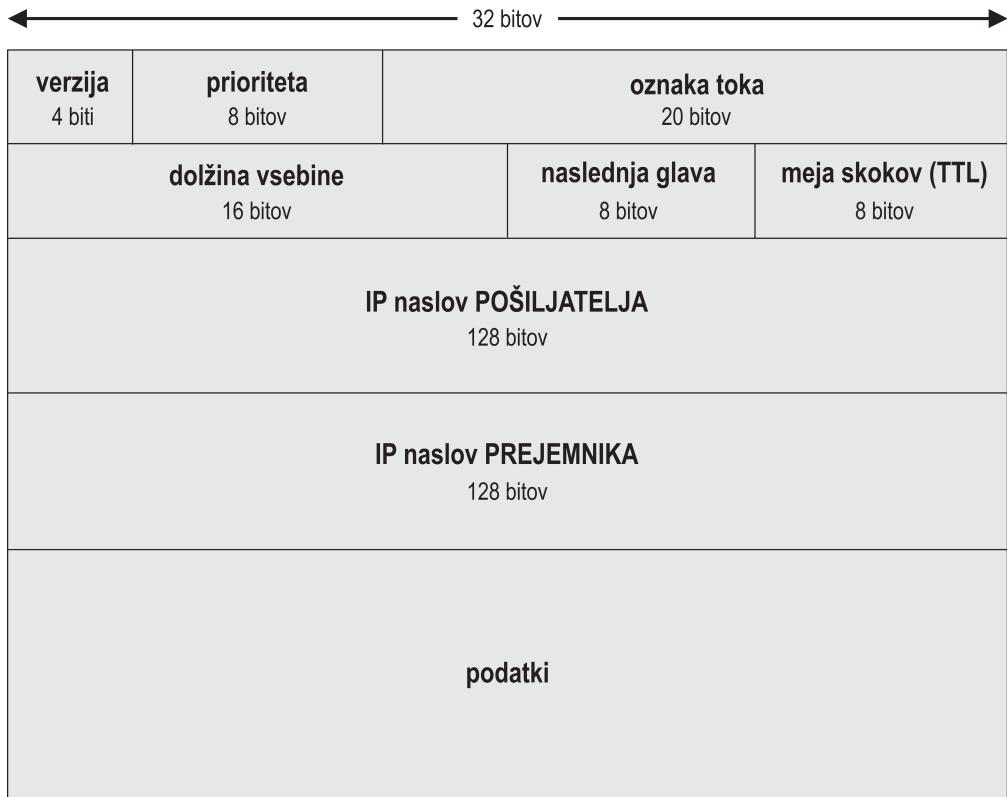
- Pomanjkanje naslovnega prostora:** Zaradi naraščanja števila priključenih naprav se je hitro izkazalo, da bo 32-bitnih naslovov sčasoma zmanjkalo. Kljub uporabi naslavljanja CIDR, zasebnih naslovnih prostorov in preslikovanja naslovov (NAT) je bilo hitro znano, da ti mehanizmi samo odlašajo neizogibno dokončno pomanjkanje 32-bitnega naslovnega prostora.
- Počasna hitrost procesiranja paketov:** Procesiranje paketov IPv4 je zamudno, saj fragmentacija zahteva svoj procesorski čas, ravno tako pa tudi branje dolge glave paketa in preračunavanje kontrolne vsote (ki je potrebno pri vsakem skoku zaradi zmanjšanja vrednosti TTL). Snovalci naslednje verzije protokola so se zato odločili ukiniti fragmentacijo (kar je dobro tudi z varnostnega stališča, saj nekateri napadi temeljijo na pošiljanju nepravilno oblikovanih fragmentiranih paketov), zmanjšati velikost glave paketa in odstraniti kontrolno vsoto (ta se že izračunava na višjih plasteh).
- Slaba podpora sodobnim aplikacijam:** Paket IPv4 ne omogoča dobre podpore različnim tipom prometa, ki potrebujejo višjo prioriteto pri prenosu (kot so tokovi multimedijskih podatkov, ki se morajo prejemniku prikazovati v realnem času). Čeprav ima paket IPv4 polje, ki določa tip prometa, se v praksi le-to ne uporablja pogosto.

Naštete težave so vodile k nadalnjemu razvoju protokola IP. V zgodnjih 90. letih je prvi poskus napredka bil protokol IST (Internet Stream Protokol), ki pa je ostal v eksperimentalni fazi in nikoli ni zaživel. Leta 1998 je organizacija IETF (Internet Engineering Task Force) standardizirala izboljšani protokol IPv6, ki je v uporabi tudi danes. Ta uvaja naslednje ključne izboljšave:

- Uporablja naslove dolžine 128 bitov in s tem naslovni prostor, ki omogoča  $3,4 \cdot 10^{38}$  možnih naslovov.
- Omogoča hitrejše procesiranje paketov, saj ima preprostejšo glavo paketa, ki ne podpira fragmentacije in ne vsebuje kontrolne vsote. Pri pošiljanju s protokolom IPv6 mora pošiljatelj poskrbeti za dovolj majhne pakete. Če so le-ti preveliki, jih usmerjevalnik ne fragmentira, temveč odgovori s sporočilom, da je paket prevelik. Glava ravno tako ne vsebuje več kontrolne vsote, ker je ta prisotna že v glavah enkapsuliranih datagramov znotraj paketa. Ker ni več potrebe po preračunavanju kontrolne vsote ob vsaki spremembi polja TTL, je procesiranje paketa hitrejše.
- Omogoča hitrejše procesiranje paketov, ker v glavi nima polja za opciske razširitve. Zaradi tega je glava vedno fiksne dolžine 40 bajtov.
- Glava paketa IPv6 ima polje za oznako prioritete vsebine in polje za oznako toka podatkov (s katerima lahko označimo podatke, ki smiselno sodijo skupaj - npr. slikovni tok prenašanega filma).

Paket protokola IPv6 je prikazan na sliki 4.3, ta pa vsebuje polja, ki so razložena v tabeli 4.5. Čeprav je dolžina glave protokola IPv6 dvakrat daljša,

lahko že na prvi pogled opazimo večjo preprostost strukture paketa IPv6, ki zagotavlja opisane performančne prednosti.



**Slika 4.3** Paket protokola IPv6

| POLJE                           | DOLŽINA  | POMEN                                                                                                    |
|---------------------------------|----------|----------------------------------------------------------------------------------------------------------|
| VERZIJA                         | 4 biti   | Verzija protokola IP (ima vrednost 6).                                                                   |
| PRIORITETA (ali RAZRED PROMETA) | 8 bitov  | podobno kot TIP STORITVE pri IPv4, oznaka prioritete za posebne pakete / aplikacije.                     |
| OZNAKA TOKA                     | 20 bitov | Oznaka skupine podatkov, kar omogoča zagotavljanje kakovosti storitve (npr. pri prenosu zvoka in slike). |
| DOLŽINA VSEBINE                 | 16 bitov | Velikost podatkov znotraj paketa (podatkov, ki sledijo glavi).                                           |
| NASLEDNJA GLAVA                 | 8 bitov  | Oznaka protokola, ki je enkapsuliran v paketu.                                                           |
| MEJA SKOKOV                     | 8 bitov  | Enako kot TTL, življenski čas paketa, ki ga vsak usmerjevalnik na poti zmanjša za 1.                     |

**Tabela 4.5** Pomen polj v paketu protokola IPv6

#### 4.4.1 Naslavljjanje IPv6

Protokol IPv6 uporablja 128-bitne naslove, kar omogoča naslavljjanje  $3,4 \cdot 10^{38}$  vmesnikov. 128-bitnega naslova običajno ne zapisujemo v binarni obliki, pač

| ★ Primer 4.10:                                                     |                                                                                                                                                          |
|--------------------------------------------------------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------|
| primer naslova IP<br>(presledki so dodani le za večjo berljivost): | 00100001 11011010 00000000 11010011<br>00000000 00000000 00000000 00000000<br>00000010 10101010 00000000 11111111<br>11111110 00101000 10011100 01011010 |
| primer naslova IP<br>(šestnajstiško)                               | 21da:00d3:0000:0000:02aa:00ff:fe28:9c5a                                                                                                                  |
| poenostavitev: izpustimo vodilne ničle                             | 21da:d3:0:0:2aa:ff:fe28:9c5a                                                                                                                             |
| poenostavitev: skrajšamo blok ničel                                | 21da:d3::2aa:ff:fe28:9c5a                                                                                                                                |

pa 8 skupinah (četverkah) šestnajstiških števil (skupaj je torej 32 šestnajstiških števil, od katerih vsako predstavlja 4 bite<sup>2</sup>,  $4 \times 32 = 128$ ), ki so ločene z dvopičji. Velikost črk (šestnajstiških znakov) pri zapisu ni pomembna, priporoča pa se uporaba malih črk. Zaradi lažjega zapisa dolgih naslovov obstajajo tudi načini njihovega krajšega zapisa, kot sledi:

- Vodilne ničle zapisanih četverk lahko izpustimo.
- Zaporedje več celotnih četverk ničel lahko zapišemo kot dvojno dvopičje (::). Če se v naslovu nahaja več skupin četverk ničel, smemo skrajšati le eno od njih, saj sicer krašanje vodi v dvoumen zapis naslova, iz katerega ni možno ugotoviti, koliko ničel je bilo odstranjenih (npr. 21da:d3:0:0:2aa:0:0:9c5a ne smemo skrajšati v 21da:d3::2aa::9c5a).
- Za večjo kompatibilnost lahko naslove IPv4 preprosto zapišemo kot naslove v obliki IPv6 tako, da jim dodamo predpono ::. Npr. naslov IPv4 193.2.72.1 lahko zapišemo kot naslov IPv6 v obliki ::193.2.72.1.

Primer takšnega postopka poenostavitev prikazuje primer 4.10.

Enako kot pri IPv4, delimo naslove v skupine glede na način pošiljanja (neposredno pošiljanje - *unicast*, razpošiljanje naročnikom - *multicast*, pošiljanje najbližnjemu - *anycast*). Pri neposrednem pošiljanju (*unicast*), ki je najpogostejša oblika naslavljanja, so 128-bitni naslovi razdeljeni v dve 64-bitni polovici:

- Prvih 64 bitov predstavlja *naslov omrežja* in ga je možno deliti naprej na dva manjša dela: najmanj 48 začetnih bitov predstavlja *usmerjevalno predpono* (angl. *routing prefix*), preostala razlika v številu bitov do 64 (torej največ 16 bitov) pa *oznako podomrežja*. Oznake podomrežja lahko poljubno določa ponudnik Internet storitev in na ta način svoj naslovni prostor, deli naprej v

<sup>2</sup> $0000_2 = 0_{16}$ ,  $0001_2 = 1_{16}$ ,  $0010_2 = 2_{16}$ ,  $0011_2 = 3_{16}$ ,  $0100_2 = 4_{16}$ ,  $0101_2 = 5_{16}$ ,  $0110_2 = 6_{16}$ ,  $0111_2 = 7_{16}$ ,  $1000_2 = 8_{16}$ ,  $1001_2 = 9_{16}$ ,  $1010_2 = a_{16}$ ,  $1011_2 = b_{16}$ ,  $1100_2 = c_{16}$ ,  $1101_2 = d_{16}$ ,  $1110_2 = e_{16}$ ,  $1111_2 = f_{16}$

manjše naslovne prostore po enakem principu kot je to izvajal pri IPv4 v razdelku 4.3.3.

- Drugih 64 bitov predstavlja *naslov vmesnika*, ki ga administrator lahko bodisi nastavi ročno, ga vmesnik prejme od strežnika avtomatsko konfiguriranje (strežnik DHCP) ali pa se avtomatsko določi na podlagi fizičnega naslova (MAC).

Posebna oblika naslovov IPv6 se uporablja za naslavljanje lokalnih vmesnikov (angl. *link-local addressing*). V tem primeru namesto prvih 64 bitov zapišemo 10-bitno predpono naslova omrežja 1111111010, ki ji sledi še 54 ničel<sup>3</sup>; naslov pa enako nadaljujemo z naslovom vmesnika kot pri običajnih naslovih. Takšno naslavljanje se uporablja zgolj za komunikacijo z drugimi lokalnimi vozlišči, saj usmerjevalniki paketov, naslovljenih na lokalne naslove, ne posredujejo naprej.

Za razliko od IPv4, protokol IPv6 ne pozna naslavljanja vseh prejemnikov z naslovom *broadcast*, ki smo ga do sedaj tvorili iz samih enic v delu naslova za naslov vmesnika. Za namene pošiljanja vsem se uporablja razpošiljanje skupinam prijavljenih prejemnikov (angl. *multicast*). IPv6 za potrebe takšnega razpošiljanja vnaprej definira nekaj naslovov, ki naslavljajo specifične prejemnike, kot so npr.: vsa lokalna vozlišča (ff01::1 in ff02::1), vsi lokalni usmerjevalniki (ff01::2, ff02::2 in ff03::2), vsi usmerjevalniki, ki izvajajo protokol RIP (ff02::9), vsi časovni strežniki (ff02::101), vsi DHCP strežniki (ff05::1:3) itd.

#### 4.4.2 Prehodni mehanizmi z IPv4 na IPv6

Ker na svetu trenutno več kot 2 milijarde ljudi<sup>4</sup> uporablja Internet, je hkraten prehod vseh uporabnikov Interneta na protokol IPv6 nemogoč. Zato se danes uporabljajo prehodni mehanizmi, ki omogočajo sobivanje obeh protokolov in delovanje Interneta s hkratno rabo obeh protokolov. Dva najbolj znana prehodna mehanizma sta **uporaba dvojnega sklada** (angl. *dual stack*) in **tuneliranje** (angl. *tunneling*).

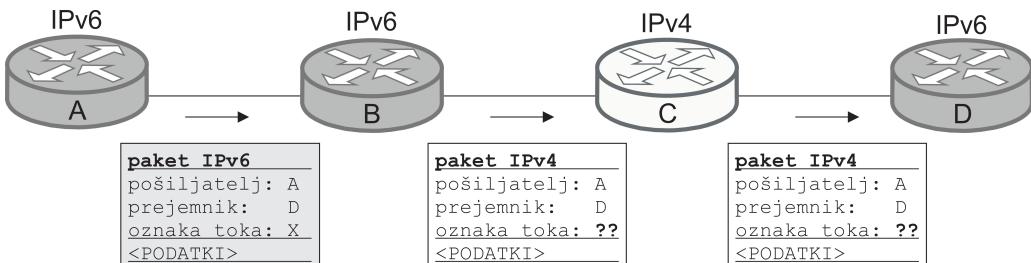
#### Dvojni sklad

Pri uporabi mehanizma **dvojnega sklada** vozlišča uporabljajo protokolni sklad, v katerem tečeta dve vzporedni implementaciji omrežnega protokola: IPv4 in IPv6. Če drugo vozlišče, s katerim želi komunicirati prvo vozlišče, že uporablja novi protokol IPv6, potem komunicirata z uporabo tega protokola; sicer se komunikacija odvije z uporabo protokola IPv4. Preden pošlje pošiljatelj prejemniku paket, mora torej ugotoviti, ali naj uporabi protokol IPv4 ali IPv6. To naredi tako, da povpraša strežnik DNS (o njem bomo govorili v poglavju 6,

<sup>3</sup>šestnajstško ta naslov omrežja zapišemo kot fe80::/64

<sup>4</sup><http://www.internetworldstats.com/>, julij 2011

★ **Primer 4.11:** Kot primer pretvorbe prometa si poglejmo usmerjevalnike A, B, C in D, od katerih A, B in D uporabljajo dvojni sklad, usmerjevalnik C pa le IPv4. Za poenostavitev prikaza so v paketih prikazana le polja z naslovom pošiljatelja, naslovom prejemnika in oznako toka (ostala polja se pri pretvorbi ohranijo). S slike vidimo, da usmerjevalnik A pošlje B paket IPv6, ki vsebuje ohranjeno vsebino vseh polj. Ker pa usmerjevalnik C uporablja IPv4, se vsebina polja za tok podatkov izgubi in ne prispe do usmerjevalnika D, čeprav ta že uporablja IPv6.



namenjen pa je pretvorbi imena strežnika v naslov IP) o številki IP prejemnika. V kolikor od strežnika DNS prejme samo naslov IPv4, ve, da prejemnik še ne uporablja protokola IPv6; v kolikor pa prejme tudi številko IPv6, ve, da lahko paket pošlje že v obliki IPv6.

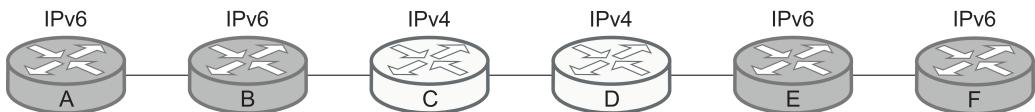
Težava pri uporabi zgornjega sistema se lahko pojavi takrat, kadar pošiljatelj, ki uporablja protokol IPv6 pošlje paket prejemniku, ki tudi uporablja protokol IPv6 (torej pošlje paket v obliki IPv6), eden od usmerjevalnikov na poti pa še ne uporablja nove verzije protokola, temveč IPv4. Takrat je potrebno paket IPv6 pretvoriti v paket IPv4, da ga vmesni usmerjevalnik lahko razume. Če primerjamo slike 4.1 in 4.3 lahko vidimo, da lahko vsebino večine polj paketa IPv6 prenesemo v obliko paketa IPv4 razen polja za oznako toka podatkov, ki je novost verzije IPv6. Zaradi tega se vsebina tega polja ob pretvorbi ne more prenesti in se izgubi, kot to prikazuje primer 4.11.

## Tuneliranje

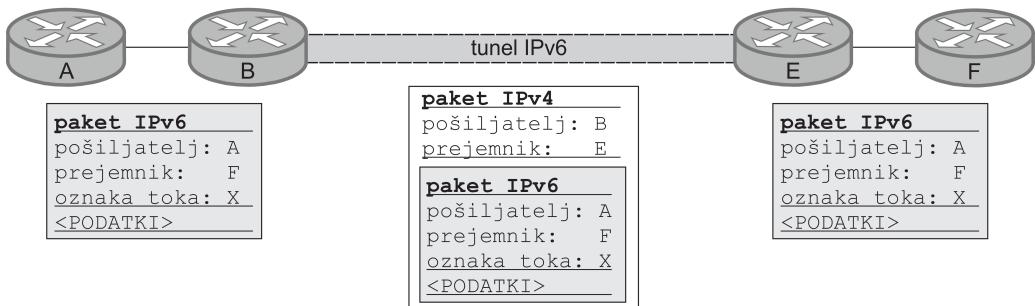
**Tuneliranje** je drugačen prehodni mehanizem, pri katerem se paketi IPv6 ne pretvarjajo v IPv4, temveč se za prenos preko starih (IPv4) vozlišč paketi IPv6 prenašajo kot podatki znotraj paketa IPv4. To pomeni, da se znotraj paketa IPv4 ne enkapsulirajo neposredno podatki transportne plasti, temveč da se v postopek doda še posreden korak enkapsulacije paketa IPv6 znotraj paketa IPv4. Delovanje postopka prikazuje primer 4.12.

★ **Primer 4.12:** Delovanje tuneliranja si lahko pogledamo na primeru, ki prikazuje vozlišča A, B, E in F (ki uporablja IPv6) in vozlišči C in D (ki uporablja IPv4). Za potrebe prenosa paketov IPv6 preko vozlišč C in D je med vozliščema B in E vzpostavljen tunel. Slike lahko vidimo, kako se prvotni paket IPv6 (ki ga je A poslal F) enkapsulira v paket IPv4 (beli zunanji kvadrat). V zunanjem paketu se ustreznno spremenita naslov pošiljatelja in prejemnika v naslove vozlišč, med katerima je vzpostavljen tunel. Ob zaključku tunela se paket IPv6 dekapsulira in v prvotni obliki posreduje naprej ciljnemu vozlišču.

Fizična slika:



Logična slika:



## 4.5 Usmerjanje

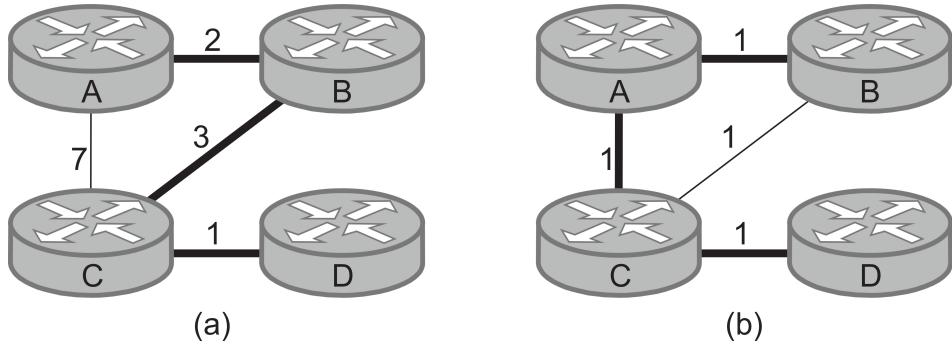
Poleg prenosa podatkov s protokolom IP in signalizacije s prenosom kontrolnih sporočil (ICMP) smo v uvodu omenili, da je ključna naloga omrežne plasti tudi izvajanje usmerjanja. V primeru iz uvoda, ki opisuje analogijo iz realnega sveta, smo omenili, da je usmerjanje v Internetu sorodno nalogam "potovalnih agencij", ki morajo poiskati pot od začetnega do končnega kraja. Naloga usmerjanja je torej **poiskati pot preko usmerjevalnikov, po kateri bo paket potoval od izvora (pošiljatelja) do cilja (prejemnika)**. Usmerjevalniki se za optimalno pot skozi omrežje dogovarjajo z uporabo **usmerjevalnih protokolov**, s katerimi si izmenjujejo medsebojna obvestila o dosegljivosti delov omrežja in na njihovi podlagi konfigurirajo svoje posredovalne tabele.

Za lažjo formalno obravnavo problema usmerjanja zato omrežje usmerjevalnikov modeliramo s teorijo grafov, kjer omrežje predstavimo kot par

$$G = \langle V, P \rangle,$$

kjer je  $V$  množica vozlišč (v našem primeru: usmerjevalnikov) in  $P$  množica povezav med njimi. Primer takšnega omrežja je podan na sliki 4.4, na kateri tudi

vidimo, da lahko posamezne povezave ovrednotimo s **cenami**, ki predstavljajo kriterij, ki ga želimo minimizirati na poti od izvora do cilja. Kot ceno na povezavah lahko usmerjevalni algoritmi upoštevajo različne kriterije, ki so lahko npr. strošek (finance), fizična razdalja, hitrost povezave, varnostna politika in druge. Pri omembi različnih usmerjevalnih protokolov bomo tudi na konkretnem primeru videli, kako vsak od njih drugače vrednoti ceno posameznih povezav.

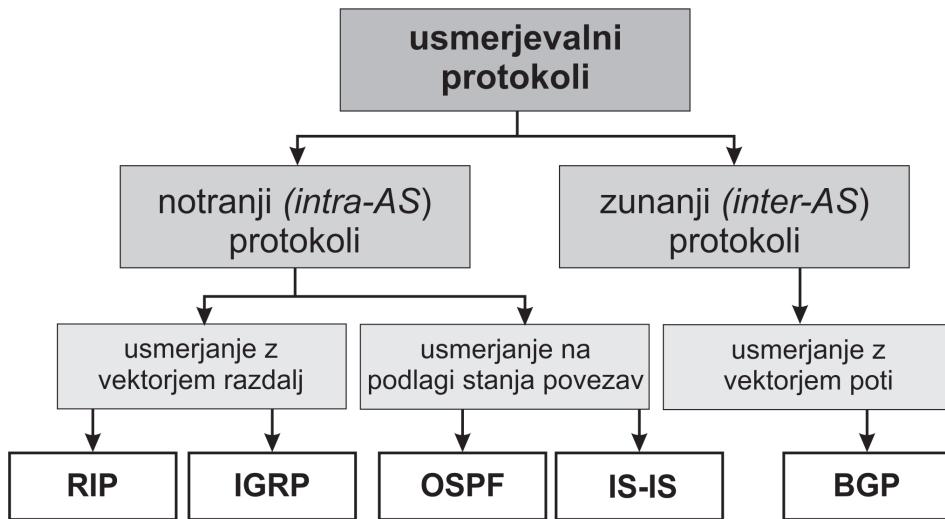


**Slika 4.4** Najkrajša pot iz vozlišča A do vseh ostalih vozlišč glede na različno vrednotenje cen povezav: (a) minimizacija poti glede na podane cene na povezavah (najkrajša cena poti od A do B znaša 2, od A do C znaša  $2 + 3 = 5$  in od A do D znaša  $2 + 3 + 1 = 6$ ); (b) minimizacija glede na število skokov (najkrajša razdalja od A do B znaša 1, od A do C 1 in od A do D  $1 + 1 = 2$ ).

Za obravnavo problema usmerjanja bomo ceno povezave med usmerjevalniki, koma  $x_1$  in  $x_2$  označili z  $c(x_1, x_2)$ ; cena celotne poti, ki se začne z usmerjevalnikom  $x_1$ , nato pa se nadaljuje z usmerjevalniki  $x_2, x_3, \dots, x_n$  pa bo enaka kar vsoti cen posameznih povezav, ki tvorijo pot:

$$c(x_1, x_2, x_3, \dots, x_n) = c(x_1, x_2) + c(x_2, x_3) + \dots + c(x_{n-1}, x_n).$$

Usmerjevalne algoritme v Internetu delimo glede na njihove lastnosti in namen, kot je prikazano na sliki 4.5, za njeno razumevanje pa je potrebno najprej definirati pojem **avtonomnega sistema** (angl. AS, *autonomous system*). Ta je definiran kot *zbirka podomrežij v Internetu, ki so pod skupno administracijo in imajo enotno definirano politiko usmerjanja*. Na osnovi mesta delovanja algoritma v omrežju lahko usmerjevalne protokole razdelimo na notranje ali *intra-AS* protokole (angl. IGP, *Interior Gateway Protocols*), ki so namenjeni usmerjanju **znotraj** posameznih avtonomnih sistemov in zunanje ali *inter-AS* protokole (angl. EGP, *Exterior Gateway Protocols*), ki so namenjeni usmerjanju **med** posameznimi avtonomnimi sistemmi. Znotraj posameznih avtonomnih sistemov se običajno uporablja isti notranji usmerjevalni protokol; med posameznimi avtonomnimi sistemmi pa se uporablja protokol, namenjen usmerjanju paketov v druge avtonomne



Slika 4.5 Delitev usmerjevalnih protokolov v Internetu

sisteme. V današnjem Internetu se kot glavni zunanji usmerjevalni protokol uporablja protokol BGP (angl. *Border Gateway Protocol*).

Notranje usmerjevalne protokole nadalje delimo v skupino **centraliziranih** in **decentraliziranih** protokolov. Centralizirani protokoli ali **protokoli za usmerjanje na podlagi stanja omrežnih povezav** (angl. *link-state routing protocols*) izračunajo optimalno pot na podlagi trenutnega stanja povezav v celotnem omrežju, ki ga zberejo pred izračunom. Za razliko od njih decentralizirani protokoli ali **protokoli za usmerjanje na podlagi vektorja razdalj** (angl. *distance vector routing protocols*) upoštevajo le podatke (vektorje razdalj do vseh usmerjevalnikov v omrežju), prejete od neposredno priključenih usmerjevalnikov, in optimalno pot posodabljajo iterativno, po vsaki izmenjavi le-teh.

V praksi lahko usmerjevalne algoritme delimo tudi glede na njihove lastnosti, kot so sposobnost prilagajanja na obremenitev povezav, upoštevanje razpoložljive pasovne širine, zakasnitve in zanesljivosti povezave. V nadaljevanju si bomo podrobneje pogledali značilnosti centraliziranih in decentraliziranih protokolov.

#### 4.5.1 Usmerjanje na podlagi stanja povezav

Pri uporabi centraliziranega usmerjevalnega protokola ima vsako vozlišče v omrežju nalogi sklepiti na stanje povezanosti med usmerjevalniki v celotnem omrežju. To naredijo tako, da si to stanje predstavijo z grafom povezanosti, v katerem nato izračunajo najkrajše poti do vseh preostalih vozlišč v omrežju (izračuna torej drevo najkrajših poti od sebe do vseh preostalih vozlišč, za kar se pogosto uporablja algoritmom Dijkstra). Iz izračunanih najkrajših poti se nato

določi posredovalna tabela za vozlišče. Primer takšnega drevesa najkrajših poti iz vozlišča A smo videli že na sliki 4.4.

Da lahko opisan postopek deluje morajo vsa vozlišča opazovati svoje povezave do sosednjih usmerjevalnikov. Periodično ali pa ob spremembah povezljivosti nato razpošljejo **sporočilo o stanju povezav** vsem ostalim vozliščem v omrežju. To sporočilo usmerjevalniki poplavijo na vse svoje povezave in ga oštevilčijo, zato da lahko prejemniki spremljajo, katera je zadnja verzija prejetega sporočila (zaradi poplavljanja lahko včasih starejša verzija sporočila prispe po novejši).

Kot primer protokola, ki usmerja na podlagi stanja povezav, omenimo **protokol OSPF** (angl. *Open Shortest Path First*). Poleg osnovne funkcionalnosti posredovanja je sposoben tudi naprednejšega usmerjanja po več različnih poteh in razpošiljanja (multicast) sporočil o stanju povezav namesto poplavljanja. Kot ceno povezave lahko upošteva čas pošiljanja, kapaciteto, razpoložljivost in zanesljivost povezave. OSPF omogoča tudi delitev omrežja v manjša področja (angl. *routing areas*), s čimer se lahko pohitri izračunavanje optimalnih poti. Zaradi zadnje opisane prednosti so protokoli za usmerjanje na podlagi stanja povezav tudi bolj skalabilni glede na velikost omrežja kot usmerjevalni protokoli iz drugih družin.

#### 4.5.2 Usmerjanje z vektorjem razdalj

Usmerjanje z vektorjem razdalj, imenovano tudi *decentralizirano usmerjanje* ali *porazdeljeno usmerjanje*, upošteva le lokalne podatke o povezavah, ki jih usmerjevalnik prejme samo od neposredno priključenih sosedov. Ti svoja sporočila pošiljajo periodično. Sporočila, ki jih imenujemo **vektorji razdalj**, vsebujejo ocene (približke) cen povezav do ostalih usmerjevalnikov v omrežju. Ker usmerjevalniki običajno niso neposredno povezani prav z vsemi ostalimi usmerjevalniki v omrežju, si morajo torej le na podlagi prejetih vektorjev razdalj od svojih sosedov, izračunati svoje ocenjene (približne) cene povezav do preostalih usmerjevalnikov, s katerimi niso neposredno povezani. To izračunavanje poteka iterativno in s številom iteracij konvergira proti optimalnim usmerjevalnim tabelam, saj se z vsako iteracijo podatek o pravi vrednosti cene vsake povezave bolj širi skozi omrežje preko usmerjevalnikov.

Usmerjevalnik  $U$  izračuna ocene najcenejše poti do nekega usmerjevalnika  $V$  tako, da po prejemu vektorjev razdalj od svojih sosedov  $\{U_1, U_2, \dots, U_k\}$  upošteva:

- dejansko ceno povezave od  $U$  do vsakega od sosedov  $U_1, U_2, \dots, U_k$  (torej  $c(U, U_1), c(U, U_2), \dots, c(U, U_k)$ ) in
- ocene (približke) cen poti do ciljnega vozlišča  $V$ , ki jih v vektorjih razdalj prejel od vsakega izmed sosednjih usmerjevalnikov; te približke ocen

označimo kot  $D(U_1, V), D(U_2, V), \dots, D(U_k, V)$ .

Usmerjevalnik v nadaljevanju izračuna, preko katerega soseda vodi najcenejša pot do ciljnega usmerjevalnika  $V$  in izbere tistega z najmanjšo ceno:

$$\begin{aligned} \operatorname{argmin} \{ & c(U, U_1) + D(U_1, V), \\ & c(U, U_2) + D(U_2, V), \\ & \dots, \\ & c(U, U_k) + D(U_k, V) \}. \end{aligned}$$

Iz zgornje enačbe lahko vidimo, da usmerjevalnik upošteva znane cene povezav svojih neposredno priključenih sosedov, ki jim prišteje ocenjene (približne) cene povezav od sosedov do ciljnega vozlišča. Primer 4.13 prikazuje eno iteracijo izračuna nove posredovalne tabele po podanem pravilu.

Posredovalne tabele v usmerjevalnikih se skozi iteracije prilagajajo situacijam v omrežju, ki vključujejo spremembe cen povezav, vzpostavitev novih povezav ali prekinitev obstoječih povezav. Zato lahko preteče več časa, preden se podatek v omrežju razširi do vseh usmerjevalnikov. Glede na hitrost širjenja informacij o spremembah ločimo dva principa, ki ju nazorno opisujeta njuni imeni v angleškem jeziku:

- **good news travel fast** (slov. *dobre novice se hitro razširijo*): podatek o znižanju cen povezav ali vzpostavitve novih povezav se skozi omrežje razširi hitro in posredovalne tabele se temu hitro prilagodijo,
- **bad news travel slow** (slov. *slabe novice se razširijo počasi*): podatek o zvišanju cen povezav ali prekinitvi povezave se širi počasi; usmerjevalniki se mu lahko prilagodijo zelo počasi, opazimo lahko pojav "štetja do neskončnosti" (angl. *count to infinity*).

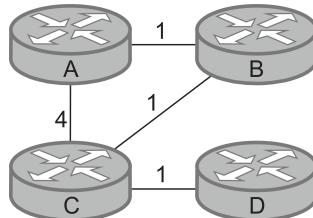
Učinkovanje obeh principov je prikazano na primerih 4.14 in 4.15.

Kot primer protokolov, ki usmerjajo z vektorji razdalj, omenimo RIP (*Routing Information Protocol*) in IGRP (*Interior Gateway Routing Protocol*). RIP kot ceno povezav uporablja število skokov (cena vsake povezave je torej 1), največja dovoljena cena pa znaša 15. Pri uporabi protokola RIP se vektorji razdalj razpošiljajo vsakih 30 s; če od soseda ne prejmemo vektorja razdalj v času 180 s, se povezava smatra za prekinjeno. Drugi protokol, IGRP, je izboljšava osnovnega protokola RIP s strani proizvajalca Cisco, ki namesto števila skokov uporablja kombinirano oceno. Ta upošteva uteženo vsoto pasovne širine, zakasnitve, obremenitve, MTU in zanesljivost povezave.

## 4.6 Varnost na omrežni plasti

Komunikacija na omrežni plasti, kot smo jo predstavili v preteklih razdelkih, poteka nevarovano. Kot takšna omogoča različne načine zlorab s strani napadalcev,

★ **Primer 4.13:** Poglejmo si omrežje štirih usmerjevalnikov, prikazanih na sliki, ki v sebi ob nekem času hranijo naslednje posredovalne tabele (zapisane so z zapisom *usmerjevalnik(ciljno vozlišče/izhodna povezava/cena povezave)*); npr. zapis B(D/C/2) pomeni, da je v posredovalni tabeli usmerjevalnika B zapis, ki pravi, da je cena poti do usmerjevalnika D preko soseda C enaka 2):



- usmerjevalnik A: A(A/-/0), A(B/C/2), A(C/C/4), A(D/C/5)
- usmerjevalnik B: B(A/A/1), B(B/-/0), B(C/C/1), B(D/C/2)
- usmerjevalnik C: C(A/A/0), C(B/B/1), C(C/-/0), C(D/D/1)
- usmerjevalnik D: D(A/C/5), D(B/C/6), D(C/C/1), D(D/-/0)

Ob nekem časovnem trenutku pošljejo vsi štirje usmerjevalniki svoje posredovalne tabele (v obliki vektorjev razdalj) svojim sosedom. Usmerjevalnik A bo na ta način prejel vektorja razdalj od sosednjih usmerjevalnikov B in C, ne pa tudi od D, ker D ni neposredni sosed usmerjevalnika A. Po prejemu vektorjev razdalj bo usmerjevalnik A izračunal nove cene razdalj do vseh ostalih usmerjevalnikov v omrežju, kot sledi:

- cena do usmerjevalnika A: 0
- cena do usmerjevalnika B:  

$$\begin{aligned} & \min(c(A,B) + D(B,B), c(A,C) + D(C,B)) = \\ & = \min(1 + 0, 4 + 1) = 1 \text{ (preko B)} \end{aligned}$$
- cena do usmerjevalnika C:  

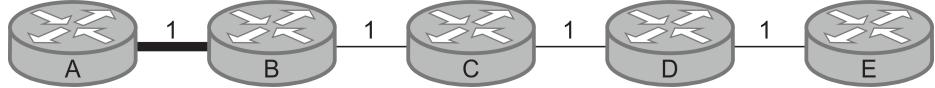
$$\begin{aligned} & \min(c(A,B) + D(B,C), c(A,C) + D(C,C)) = \\ & = \min(1 + 1, 4 + 0) = 2 \text{ (preko B)} \end{aligned}$$
- cena do usmerjevalnika D:  

$$\begin{aligned} & \min(c(A,B) + D(B,D), c(A,C) + D(C,D)) = \\ & = \min(1 + 2, 4 + 1) = 3 \text{ (preko B)} \end{aligned}$$

Po tej iteraciji si usmerjevalnik A torej shrani naslednjo novo posredovalno tabelo: A(A/-/0), A(B/B/1), A(C/B/2), A(D/B/3).

Za vajo izračunaj še nove posredovalne tabele usmerjevalnikov B, C in D.

## ★ Primer 4.14:



Kot primer si poglejmo omrežje petih usmerjevalnikov, ki so povezani zaporedno, cene vseh povezav pa so enake 1. Opazujmo, kako se bo spremenjal podatek o najkrajši poti do usmerjevalnika A v posredovalnih tabelah usmerjevalnikov B, C, D in E, če je povezava med A in B na začetku prekinjena, nato pa se vzpostavi. Po vzpostavitvi bo v prvi iteraciji usmerjevalnik B prejel vektor razdalj od A in si na njegovi podlagi shranil zapis  $B(A/A/1)$ . V drugi iteraciji bo B ta zapis v svojem vektorju razdalj posredoval sosedoma (usmerjevalnikoma A in C), zato bo usmerjevalnik C lahko v svojo posredovalno tabelo shranil  $C(A/B/2)$ . Scenarij se bo nadaljeval po enakem principu, razdalje pa se bodo v štirih iteracijah razširile po celiem omrežju (podani so zapisi *usmerjevalnik(ciljno vozlišče/izhodna povezava/cena povezave)* za usmerjevalnike B, C, D in E):

- začetek:  $B(A/-/\infty)$ ,  $C(A/-/\infty)$ ,  $D(A/-/\infty)$ ,  $E(A/-/\infty)$ ,
- 1. iteracija:  $B(A/A/1)$ ,  $C(A/-/\infty)$ ,  $D(A/-/\infty)$ ,  $E(A/-/\infty)$ ,
- 2. iteracija:  $B(A/A/1)$ ,  $C(A/B/2)$ ,  $D(A/-/\infty)$ ,  $E(A/-/\infty)$ ,
- 3. iteracija:  $B(A/A/1)$ ,  $C(A/B/2)$ ,  $D(A/C/3)$ ,  $E(A/-/\infty)$ ,
- 4. iteracija:  $B(A/A/1)$ ,  $C(A/B/2)$ ,  $D(A/C/3)$ ,  $E(A/D/4)$ .

Zaradi hitrega širjenja podatka po omrežju pravimo, da gre za princip *hitrega širjenja dobrej novic*.

kot so npr.:

1. ponarejanje izvornih ali ponornih naslovov (*IP spoofing*),
2. prisluškovanje (nepooblaščeno branje vsebine paketov),
3. ponarejanje vsebine podatkov v paketih,
4. ponavljanje komunikacije itd.

V splošnem si prizadevamo uporabljati varnostne mehanizme na vseh plasti komunikacijskega modela. Za potrebe varnosti na omrežni plasti pogosto uporabljamo nabor protokolov **IPSec**, v kateri sta ključna protokol AH (*Authentication Header*) in ESP (*Encapsulating Security Payload*). Podrobnosti delovanja teh dveh protokolov presegajo obseg učbenika, povejmo pa, da IPSec omogoča enkripcijo (zakrivanje) in dekripcijo komunikacije, zagotavljanje integritete (nespreminjanja) podatkov, avtentifikacijo udeležencev in dogovor o uporabljenih metodah za kriptografijo, ki so podrobneje opisane v poglavju 7.

## ★ Primer 4.15:



Kot primer si poglejmo omrežje petih usmerjevalnikov, ki so povezani zaporedno, cene vseh povezav pa so enake 1. Denimo, da so na začetku vzpostavljene optimalne posredovalne tabele (kot je podano spodaj), povezava med A in B pa naj se nato prekine.

Po prekinitvi bo usmerjevalnik B prejel vektor razdalj samo še od usmerjevalnika C (ne pa tudi od A). Ta o prekinitvi povezave med A in B še ni bil obveščen in bo usmerjevalniku B sporočil njegovo ocenjeno ceno do A:  $A(A/B/2)$ . Tej ocenjeni ceni bo B pri izračunu nove posredovalne tabele prištel ceno razdalje med B in C in izračunal novo pot do A in si zapisal  $B(A/C/3)$ . Že v tej točki lahko opazimo anomalijo pri izračunu nove posredovalne tabele usmerjevalnika B, ki je nastala zaradi počasnega in iterativnega razširjanja podatkov skozi omrežje.

V naslednji iteraciji bo usmerjevalnik B izračunano ceno  $B(A/C/3)$  poslal svojim sosedom (pravzaprav edinemu sosedu - C), ki bo novo ceno poti do usmerjevalnika A izračunal kot minimum cen preko usmerjevalnika B ( $1 + 3$ ) ali usmerjevalnika D ( $1 + 3$ ). Ker sta obe vsoti enaki, za nas pa je pomembna le sprememba cene (nova cena znaša 4), denimo, da bo v novo posredovalno tabelo shranil  $C(A/B/4)$  (lahko pa bi tudi  $C(A/D/4)$ ). Naslednje iteracije potekajo, kot sledi:

- začetek:  $B(A/A/1)$ ,  $C(A/B/2)$ ,  $D(A/C/3)$ ,  $E(A/D/4)$ ,
- 1. iteracija:  $B(A/C/3)$ ,  $C(A/B/2)$ ,  $D(A/C/3)$ ,  $E(A/D/4)$ ,
- 2. iteracija:  $B(A/C/3)$ ,  $C(A/B/4)$ ,  $D(A/C/3)$ ,  $E(A/D/4)$ ,
- 3. iteracija:  $B(A/C/5)$ ,  $C(A/B/4)$ ,  $D(A/C/5)$ ,  $E(A/D/4)$ ,
- 4. iteracija:  $B(A/A/5)$ ,  $C(A/B/6)$ ,  $D(A/C/5)$ ,  $E(A/D/6)$ .

Ker se cene povezav v posredovalnih tabelah počasi povečujejo, pravimo, da gre za princip *počasnega širjenja slabih novic*, konkreten pojav koračnega povečevanja cen do nedosegljivega usmerjevalnika pa imenujemo *šteje do neskončnosti*.

# 5 Transportna plast

Transportna ali prenosna plast leži v komunikacijskem modelu nad omrežno plastjo in nudi svoje storitve aplikacijski plasti. Protokoli transportne plasti zagotavljajo prenos (transport) aplikacijskih sporočil med aplikacijama v dveh oddaljenih računalnikih na transparenten način – to pomeni, da programerju ni potrebno skrbeti za podrobnosti omrežja. V Internetu na transportni plasti delujeta protokola TCP in UDP, ki se po lastnostih močno razlikujeta. Pri pregledu protokolov bomo opazili, da je protokol TCP eden izmed bolj kompleksnih protokolov, saj zagotavlja mehanizme zanesljive dostave, ki jih nižje plasti v Internetu ne implementirajo.

## 5.1 Storitve transportne plasti

Protokoli transportne plasti skrbijo za izvedbo *logične komunikacije* med aplikacijskimi procesi. To pomeni, da dva aplikacijska procesa v dveh oddaljenih računalnikih lahko med seboj komunicirata s povezavo, ki navidezno ne rešuje problemov, ki ga že rešujejo nižje plasti, kot so npr. prenos po posameznih povezavah in iskanje poti skozi omrežje. Omenjena logična povezava je izvedena tako, da pošiljatelj aplikacijsko sporočilo razbije v t.i. *segmente*<sup>1</sup> in jih nadalje predá v enkapsulacijo protokolu omrežne plasti. Na drugi strani prejemnik izvede obratno dejanje: dekapsulira segmente iz paketov, jih združi v celovita sporočila in le-ta nato predá aplikacijski plasti. Ključna razlika med omrežno in transportno plastjo je torej ta, da medtem ko omrežna plast skrbi za povezovanje *končnih sistemov*, transportna plast skrbi za povezovanje *aplikacijskih procesov* v teh oddaljenih končnih sistemih.

Iz dejstva, da aplikacijska plast uporablja storitve transportnih protokolov sledi, da je kakovost prenosa aplikacijskih sporočil omejena in močno odvisna od načina delovanja transportnega protokola. Vsak transportni protokol lahko namreč zagotavlja svoj nabor storitev, takšno razliko bomo tudi opazili med protokoloma TCP in UDP. Medtem, ko protokol TCP med drugim nudi zanesljivo, povezavno storitev, nadzor zasičenja in nadzor pretoka, nudi protokol UDP nezanesljivo in nepovezavno storitev. Odsotnost mehanizmov zanesljive komunikacije pri

---

<sup>1</sup>Omenimo, da protokolarne enote protokola UDP raje imenujemo datagrami in ne segmenti. Uporabljamo torej besedni zvezi "segment TCP" in "datagram UDP".

protokolu UDP ne pomeni, da je ta protokol manj vreden, saj ima druge prednosti v okoljih, ker mehanizmov zanesljivosti ne potrebujemo.

## 5.2 Komunikacija med aplikacijo in omrežjem

Če želimo povezati dva oddaljena procesa, moramo najprej zagotoviti način naslavljanja med njima (na enak način kot smo zagotoviti naslavljanje med končnimi sistemi na omrežni plasti, ki se v Internetu izvaja z naslovi IP). Za ta namen ima vsaka aplikacija svojo *vstopno točko*, ki jo imenujemo *vtič* (angl. *socket*). Vtič je podatek, ki je sestavljen iz para komponent: *naslova naprave* in *naslova aplikacijskega procesa*, ki ga imenujemo tudi *vrata* (angl. *port*). Če na nekem končnem sistemu teče več procesov, je torej naravno, da ima vsak od njih svoj vtič, preko katerega proces sprejema in oddaja sporočila.

Ker vemo, da je prva komponenta vtiča (naslov naprave) že zapisana v paket omrežne plasti, lahko sklepamo, da bo druga komponenta (številka vrat) morala biti zapisana v protokolarne enote transportnih protokolov. Vsak transportni segment hrani torej podatke o *izvornih vratih* in *ciljnih vratih*. Izvorna vrata so oznaka procesa, ki pošilja, in se uporabljajo za odgovor temu procesu, ciljna vrata pa so oznaka procesa na ciljni strani. Videli bomo, da sta oba podatka pri protokolih UDP in TCP dolga 16 bitov.

V Internetu so nekatere številke vrat standardizirane (imenujemo jih angl. *well-known ports*), kot so npr. številke vrat, na katerih delujejo naslednji aplikacijski protokoli:

- spletni strežnik (HTTP): 80,
- poštni strežnik (SMTP): 25,
- imenski strežnik (DNS): 53,
- oddaljen dostop (telnet): 23,
- pogovorni strežnik (IRC): 194.

Znane aplikacije uporabljajo številke vrat v obsegu 0-1024<sup>2</sup>, medtem ko višje številke vrat aplikacije pretežno izbirajo naključno kot svoje izvorne številke vrat, s katerih pošiljajo.

 **Varnostni pomislek:** Kadar napadalca zanima, kateri aplikacijski procesi tečejo na nekem končnem sistemu, lahko izvede napad **portscan**. V okviru tega napada pošlje niz aplikacijskih poročil, od katerih vsakega nameni drugim ciljnim vratom prejemnika. Na osnovi številk vrat, s katerih dobi odgovor prejemnika, lahko sklepa, katera programska oprema teče na njem. V nadaljevanju lahko napad nadaljuje z iskanjem šibkosti te programske opreme in jo zlorabi (npr. za nepooblaščen dostop do sistema, onesposobitev sistema ipd.).

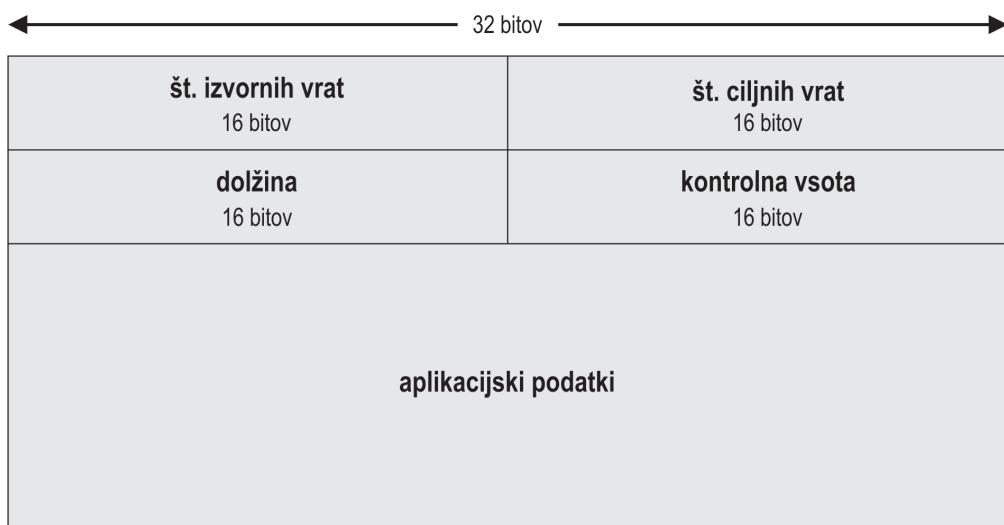
<sup>2</sup>Celoten seznam si lahko ogledamo na straneh <http://www.iana.org>.

### 5.3 Protokol UDP

Prvi transportni protokol, ki ga bomo omenili - *User Datagram Protocol (UDP)*, je preprost. Podobno kot protokol IP na omrežni plasti, nudi le storitev prenosa po najboljših zmožnostih (angl. *best-effort*). Pri prenosu ne ugotavlja, kateri datagrami so se morda izgubili, ne skrbi za njihov vrstni red, je nepovezaven (nima faze vzpostavljanja povezave), ne hrani podatkov o stanju povezave in nima dodatnih storitev (kot sta npr. nadzor pretoka ali nadzor zasičenja).

Kljub tem pomanjkljivostim pa ravno njegova preprostost zagotavlja to, da je z njim možno prenašati podatke hitro. Njegova lahka in minimalistična implementacija, ki ima le 8 bajtov dolgo glavo datagrama, omogoča tudi to, da ga je možno implementirati z zelo malo programske kode in je zato primeren tudi za implementacijo v strojni opremi (uporablja ga npr. aplikacijski protokol BOOTP, ki je tudi implementiran v strojni opremi in se uporablja za zagon operacijskega sistema računalnika preko omrežja). Protokol UDP uporabljam torej v okoljih, v katerih lahko toleriramo izgube in je pretežno pomembna le hitrost pošiljanja; takšne aplikacije so npr. prenos multimedije, protokoli DNS, SNMP (za upravljanje) in usmerjevalni protokoli. V kolikor bi aplikacija, ki uporablja protokol UDP, želela skrbeti tudi za zanesljivo dostavo, mora za le-to torej poskrbeti sama.

Slika 5.1 prikazuje strukturo datagrama UDP, ki odraža opisano preprostost. V glavi datagrama vidimo le polja za izvorna in ciljna vrata, dolžino datagrama (vključno z glavo) in internetno kontrolno vsoto, vsebina datagrama pa vsebuje enkapsulirane aplikacijske podatke. Polja so podrobneje razložena v tabeli 5.1.



**Slika 5.1** Datagram protokola UDP

| POLJE             | DOLŽINA              | POMEN                                                                                                                                                                                                            |
|-------------------|----------------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| ŠT. IZVORNIH VRAT | 16 bitov             | Številka izvornih vrat, ki je običajno najključno določena z vrednostjo $\geq 1024$ in potrebna za odgovor pošiljaljevemu procesu.                                                                               |
| ŠT. CILJNIH VRAT  | 16 bitov             | Številka, potrebna za naslavljjanje procesa na prejemnikovi strani. Za določene (strežniške) aplikacije obstajajo ustaljene (standardizirane) številke vrat.                                                     |
| DOLŽINA           | 16 bitov             | Določa skupno dolžino glave in enkapsuliranih aplikacijskih podatkov.                                                                                                                                            |
| KONTROLNA VSOTA   | 16 bitov             | Kontrolna vsota, izračunana z algoritmom <i>Internet Checksum</i> , ki je izračunana nad podatki iz glave paketa IP (naslov pošiljalja, naslov prejemnika, protokol, dolžina) in vsemi podatki iz UDP datagrama. |
| APL. PODATKI      | spremenljiva dolžina | Enkapsulirani podatki aplikacijske plasti.                                                                                                                                                                       |

Tabela 5.1 Pomen polj v datagramu protokola UDP

★ **Primer 5.1:** Primer izračuna internetne kontrolne vsote

|                                                |                                   |
|------------------------------------------------|-----------------------------------|
| prva 16-bitna beseda                           | 1 1 1 0 0 1 1 0 0 1 1 0 0 1 1 0   |
| druga 16-bitna beseda                          | 1 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1   |
| začasna vsota                                  | 1 1 0 1 1 1 0 1 1 1 0 1 1 1 0 1 1 |
| prenos odstranimo                              | →                                 |
| prenos prištejemo                              | 1 0 1 1 1 0 1 1 1 0 1 1 1 1 0 0   |
| KONTROLNA VSOTA (eniški komplement)            | 0 1 0 0 0 1 0 0 0 1 0 0 0 0 1 1   |
| ↓ prenos k prejemniku ↓                        |                                   |
| prejemnik ponovi izračun vsote 16-bitnih besed | 1 0 1 1 1 0 1 1 1 0 1 1 1 1 0 0   |
| prejemnik prišteje prejeto kontrolno vsoto     | + 0 1 0 0 0 1 0 0 0 1 0 0 0 0 1 1 |
| rezultat (pravilen sprejem)                    | 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1   |

Internetno kontrolno vsoto smo srečali že pri protokolu IP, nismo pa še podrobneje pojasnili, kako deluje. Pošiljalj izračuna vrednost kontrolne vsote tako, da sporočilo razdeli na 16 bitne besede, te nato sešteje in morebitni dodatni najvišji bit, ki je posledica prenosa pri seštevanju, odstrani ter ga prišteje k najmanj pomembnemu bitu. Eniški komplement (zamenjavi ničel za enice in enic za ničle) dobljene vsote predstavlja končno vrednost kontrolne vsote. Prejemnik kontrolno vsoto uporabi tako, da ponovi seštevanje 16 bitnih besed po enakem postopku, kot je to naredil pošiljalj, tej vsoti pa prišteje še vrednost kontrolne vsote. Če kot rezultat dobi same enice, to interpretira, kot da je kontrolna vsota skladna z vsebino sporočila. Primer izračuna kontrolne vsote je prikazan s primerom 5.1.

Vidimo lahko, da je internetna kontrolna vsota razmeroma preprost algoritem

za zaznavanje napak. Hitro lahko ugotovimo tudi, da lahko algoritem preprosto tudi zlorabimo, saj lahko isto vrednost kontrolne vsote dobimo tudi, če v sporočilu popolnoma zamenjamo vrstni red 16-bitnih besed. Kljub temu pa je postopek opisanega seštevanja računsko hiter in preprost. Ker nimamo zagotovila, da protokoli na nižjih plasteh uporabljajo zaznavanje napak, so se načrtovalci protokola UDP odločili uporabljati kontrolno vsoto tudi na transportni plasti. Poudariti je potrebno, da lahko do napak pride tudi pri hranjenju datagrama v delovnem spominu usmerjevalnika in ne nujno pri prenosu po komunikacijskem mediju, zaradi česar je mehanizem za zaznavanje napak še dodatno dobrodošel. Dodatno lahko poudarimo, da kontrolna vsota v datagramu transportne plasti simbolično predstavlja tudi preverjanje pravilnosti prenosa podatkov med izvornim in ciljnim procesom in ne preverjanju pravilnosti prenosa datagrama po posameznih povezavah.

## 5.4 Zanesljiva dostava

Ker noben od protokolov, ki smo jih spoznali (na nižjih in na transportni plasti), ne rešuje problema zanesljive dostave, si želimo protokola, ki zna nadoknaditi storitve, ki jih internetni model storitev ne zagotavlja. To pomeni, da želimo opnašati delovanje zanesljivega komunikacijskega kanala z uporabo dejanskega – nezanesljivega. Pri zanesljivem prenosu želimo skrbiti za to, da se podatki ne okvarjajo (zamenjave vrednosti bitov), ne izgubljajo in da so dostavljeni v pravilnem vrstnem redu.

Kasneje bomo spoznali protokol TCP, ki je povezaven in zanesljiv protokol. Poleg zanesljivosti pri prenosu skrbi tudi za nadzor pretoka (angl. *flow control*) in kontrolo zasičenja (angl. *congestion control*). Preden pa pogledamo strukturo datagrama tega protokola, se posvetimo študiju njegovih principov delovanja, ki nam bodo nadalje pomagali razumeti njegovo implementacijo. S tem namenom bomo sprva modelirali preprost protokol, ki mu bomo nato postopoma dodajali funkcionalnosti, kot jih ima protokol TCP. Začeli bomo s funkcionalnostjo pošiljanja in prejemanja podatkov, to bomo pa nato nadgradili z različnimi mehanizmi potrjevanja in ponovnega pošiljanja.

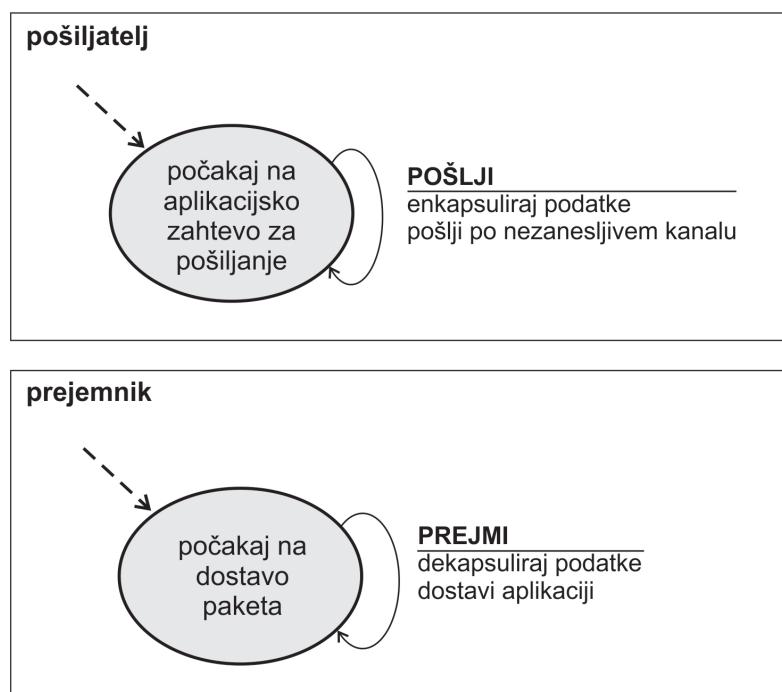
Za modeliranje delovanja protokola bomo uporabili končne avtomate. Ti so predstavljeni z množico stanj, v katerih protokol čaka (npr. čakanje na zahtevo, čakanje na potrditev) in povezav, ki opisujejo prehode med stanji. Ob vsaki povezavi bo podan pogoj za izvedbo povezave (prikazan nad vodoravno črto) in aktivnosti, ki jih protokol izvede ob prehodu (prikazana pod vodoravno črto).

Čeprav smo datagrame transportne plasti poimenovali *segmenti*, bomo v komunikacijskih diagramih v naslednjih razdelkih namesto *segmenta* pogosto uporabili izraz *paket*. Paket omrežne plasti je namreč tisti, ki vsebuje naslova

pošiljatelja in prejemnika, ta podatka pa sta torej pomembna, če opazujemo potek komunikacije med dvema končnima sistemoma.

#### 5.4.1 Pošiljanje in prejemanje podatkov

Končni avtomat procesov, ki skrbita zgolj za pošiljanje in prejemanje aplikacijskih podatkov, prikazuje slika 5.2. S slike vidimo, da pošiljatelj v svojem edinem stanju čaka na aplikacijsko zahtevo za pošiljanje. Ko ta pride, pošiljatelj enkapsulira aplikacijske podatke v datagrame transportne plasti in jih pošlje po nezanesljivem kanalu. Po drugi strani je privzeto stanje prejemnika čakanje na dostavo datagrama. Kadar ga prejemnik sprejme, iz njega dekapsulira aplikacijske podatke in jih dostavi aplikaciji.

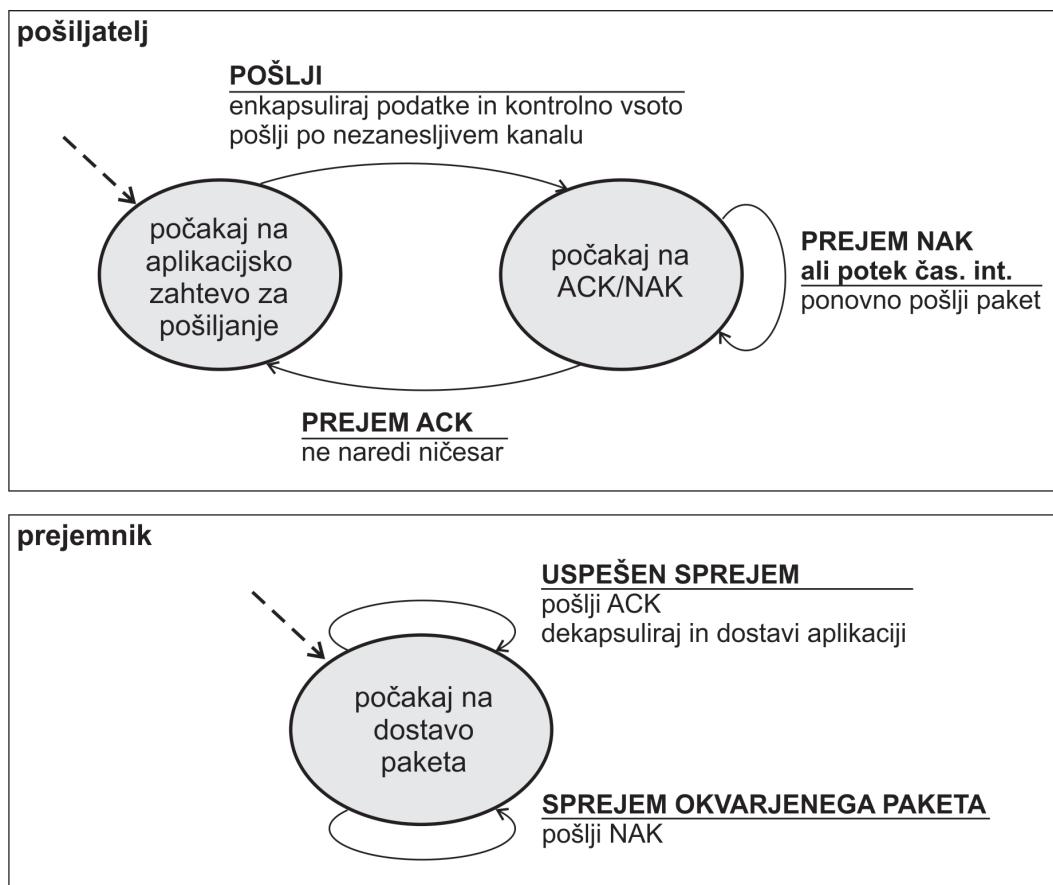


**Slika 5.2** Končni avtomat pošiljatelja in prejemnika, ki omogoča zgolj pošiljanje in prejemanje aplikacijskih podatkov

#### 5.4.2 Potrjevanje in ponovno pošiljanje

Pri pošiljanju paketov se lahko zgodi, da se vsebina okvari. Ta problem znamo že zaznati, in sicer z algoritmi za zaznavanje napak (protokoli IP, UDP in TCP za ta namen uporabljajo internetno kontrolno vsoto). S tem namenom v naslednjem koraku razširimo končna avtomata s slike 5.2 s potrjevanjem. Potrjevanje zasnujmo tako, da prejemnika zadolžimo, naj ob prejemu vsakega paketa najprej

preveri njegovo pravilnost, nato pa naj pravilno sprejete potrdi s potrditvijo ACK (angl. ACKnowledgment), o nepravilno sprejetih pa naj pošiljatelja obvesti z negativno potrditvijo NAK (angl. Negative AcKnowledgment). Kadar pošiljatelj prejme prejemnikov NAK, naj ta isti paket pošlje ponovno, v upanju, da bo v drugem poskusu sprejet pravilno.



**Slika 5.3** Končni avtomat pošiljatelja in prejemnika, ki poleg pošiljanja in sprejemanja izvajata tudi neposredno potrjevanje paketov s pozitivno potrditvijo (ACK) in negativno potrditvijo (NAK). Za zaznavanje izgubljenih paketov pošiljatelj uporablja tudi časovno kontrolo.

Končna avtomata opisanih pošiljatelja in prejemnika prikazuje slika 5.3. Slika prikazuje, pošiljatelj privzeto čaka v začetnem stanju na aplikacijsko zahtevo za prenos podatkov. Kadar ta pride, enkapsulira podatke, le-tem pa pripne tudi kontrolno vsoto, ki je potrebna za zaznavanje napak v podatkih, in datagram pošlje po nezanesljivem kanalu. Od tu naprej preide v stanje čakanja na bodisi pozitivno (ACK) ali pa negativno potrditev (NAK). V primeru, če od prejemnika prejme negativno potrditev NAK, pošiljatelj ponovi pošiljanje paketa in ostane

v stanju čakanja na potrditev (ta cikel se lahko tudi večkrat ponovi). Kadar pošiljatelj končno prejme pozitivno potrditev ACK, se lahko vrne v začetno stanje čakanja na aplikacijsko zahtevo.

Avtomat prejemnika že na prvi pogled izgleda bolj preprosto, saj ima samo eno stanje. V njem prejemnik čaka na dostavo paketa in glede na njegov pravilni ali nepravilni sprejem odgovori s potrditvama ACK ali NAK. Le v primeru pravilnega sprejema paketa preda dekapsulirane podatke aplikaciji, sicer pa jih zavrže.

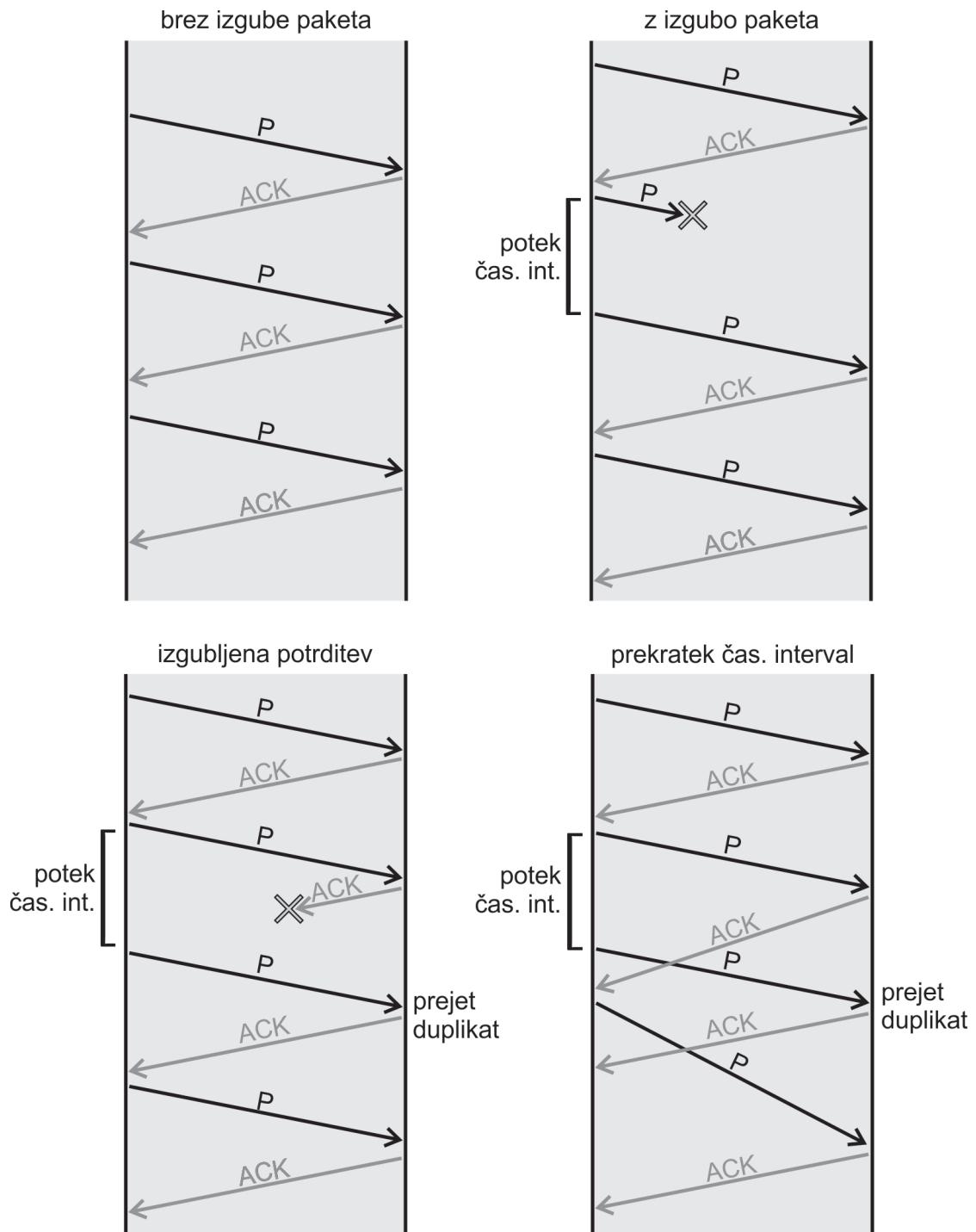
Iz delovanja pošiljatelja lahko vidimo, da dokler ne prejme pozitivne potrditve tekočega paketa, ne more sprejeti nove aplikacijske zahteve po transportu in pošiljati naslednjega paketa. Takšnemu načinu potrjevanja paketov pravimo tudi **sprotno potrjevanje**, protokolu pa, da deluje po principu *ustavi in čakaj* (angl. *stop&wait*). V nadaljnjih poglavijih bomo poskusili to omejitev odpraviti.

#### 5.4.3 Obvladovanje izgube segmentov

Poleg okvare paketov se pri prenosu lahko priperi tudi njihova izguba. Razlogi za izgube so lahko številni: napačen sprejem v napravah na poti, okvara pomnilnika, napake pri konfiguraciji usmerjanja, prekinitve povezav in drugi. Tudi v primeru izgube paketa moramo poskrbeti, da jo bo pošiljatelj lahko zaznal in ponovil pošiljanje.

Nadgradimo dosedanji protokol tako, da pošiljatelja opremimo z mehanizmom **časovne kontrole** pošiljanja paketov. S časovno kontrolo merimo časovni interval določene dolžine in če se le-ta izteče, pošiljatelj pa v tem času ne prejme potrditve prejemnika, lahko pošiljatelj sklepa, da se je poslani paket izgubil in da ga prejemnik ni prejel. Oglejmo si različne možne scenarije uporabe časovne kontrole, ki so prikazani na sliki 5.4. Zaradi večje preglednosti na narisanih scenarijih prikazujemo samo časovne kontrole, ki potečejo.

Prvi scenarij (zgoraj levo) prikazuje uspešno pošiljanje paketov in prejemanj potrditev, ki jih pošiljatelj prejme pred potekom časovne kontrole. Drugi scenarij (zgoraj desno) opisuje prej opisani primer, ko se poslani paket izgubi (prenos paketa je prekinjen s križcem). S slike vidimo, da po preteku časovnega intervala pošiljatelj ponovi pošiljanje paketa, zanj pa tokrat uspešno prejme potrditev. Sekvenca pošiljanj in potrjevanj se zato lahko nadaljuje. Tretji scenarij (spodaj levo) prikazuje, da se potek časovnega intervala lahko zgodi tudi v primeru, ko se izgubi potrditev (tudi izguba potrditve je možen dogodek, saj je pri protokolu TCP potrditev ravno tako segment protokola TCP). Ker zaradi poteka časovne kontrole pošiljatelj ponovi pošiljanje paketa, prejemnik prejme podvojeni paket. To težavo bomo pri nadaljnji konstrukciji protokola odpravili z uvedbo številčenja paketov, na podlagi katerega bo prejemnik lahko pakete s ponovljenimi številkami zavrgel. Četrти scenarij (spodaj desno) prikazuje pomen nastavitev pravilne dolžine časovnega intervala. Če je ta prekratek, kot prikazuje slika, lahko pošiljatelj



**Slika 5.4** Različni scenariji, do katerih lahko pride pri izgubi paketov ali potrditev ter uporabi časovne kontrole.

“neučakano” sklepa, da se je paket izgubil, in ponovi pošiljanje paketa, čeprav bo potrditev za prvo pošiljanje še prejel. Tudi v tem scenariju prejemnik posledično prejme isti paket dvakrat, dodatno pa tudi pošiljatelj prejme dvojno potrditev za isti paket (kar pa ne predstavlja težave). Omenimo lahko še scenarij (ki ni ilustriran), ko je nastavljen časovni interval predolg. V tem primeru komunikacija poteka enako kot v drugem scenariju, le s to razliko, da pošiljatelj dlje časa čaka na potek intervala in je s tem komunikacijski kanal slabše izkoriščen.

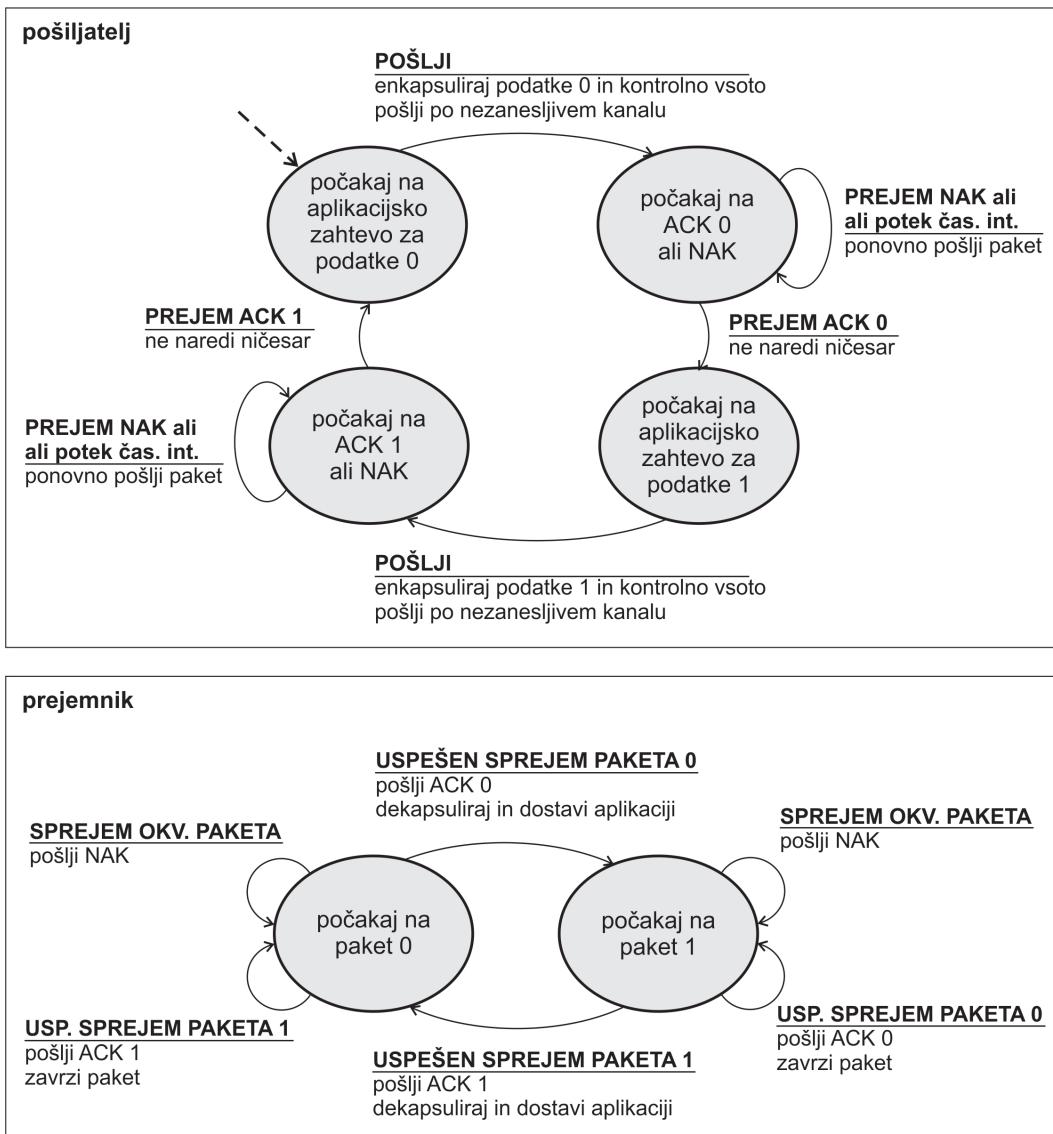
Slika 5.5 prikazuje končni avtomat protokola, ki zna reševati tudi težave z izgubljenimi paketi in potrditvami. Za potrebe zaznavanja podvojenih paketov uvedemo številčenje paketov s izmenjujočima zaporednima številkama 0 in 1. Dve številki paketov v našem primeru popolnoma zadoščata, saj še vedno uporabljamo sprotno potrjevanje oziroma protokol, ki deluje po principu *ustavi in čakaj*. Ob normalnem (brezizgubnem) pošiljanju lahko torej prejemnik pričakuje, da bo dobival pakete, oštevilčene v zaporedju 0, 1, 0, 1 itd. V primeru, da se zgodita tretji (izguba potrditve) ali četrti scenarij (prekratek časovni interval) s slike 5.4, bo prejemnik podvojen paket zaznal z dvakratnim prejemom iste številke (prejel bo npr. zaporedje 0, 1, 1, ...) ter bo lahko duplikat zavrgel.

Vidimo lahko, da je končni avtomat pošiljatelja simetričen po diagonali, razlikuje se le v številki paketa. Pošiljatelj v začetnem stanju čaka na aplikacijsko zahtevo. Kadar ta pride, oštevilči prvi paket s številko 0, izračuna kontrolo vsoto in pošlje enkapsulirane podatke. V fazi čakanja na potrditev ponovno pošlje isti paket, če od prejemnika prejme NAK, okvarjeno potrditev ali če poteče časovni interval. V primeru, da uspešno prejme ACK, preide v fazo čakanja na naslednje podatke za pošiljanje, ki jih bo oštevilčil s številko 1 ter simetrično nadaljeval po diagramu.

Prejemnikov diagram je ravno tako simetričen. V prvem od obeh stanj prejemnik čaka na paket, ki ima številko 0, v drugem pa na paket, ki ima številko 1. Vsakič, kadar prejemnik prejme neokvarjen paket z ustrezno številko, dekapsulira podatke, jih dostavi aplikaciji in vrne pošiljatelju ustrezno potrditev. V primeru, da v vsakem čakajočem stanju prejemnik prejme okvarjen paket, odgovori z NAK. V primeru, da v čakajočem stanju prejemnik prejme paket, ki ima drugačno številko od pričakovane, lahko sklepa, da je to ponovjen prejšnji paket – tega sicer potrdi, a vsebino zavrže. Na to, da gre resnično za prejšnji paket, lahko prejemnik sklepa zato, ker uporabljamo sprotno potrjevanje. Le v tem primeru se nam namreč ne more zgoditi, da z zakasnitvijo prejmemmo paket, ki je starejši od prejšnjega.

#### 5.4.4 Neposredno in posredno potrjevanje

Potrjevanju, ki ga izvajamo s pozitivnim (ACK) in negativnim (NAK) potrjevanjem, pravimo tudi **neposredno potrjevanje**. Izkaže se, da lahko ta postopek poenostavimo v **posredno potrjevanje**, ki uporablja samo pozitivne potrditve. S



**Slika 5.5** Končni avtomat pošiljatelja in prejemnika, ki izvajata sprotno potrjevanje s številčenjem paketom. Na podlagi ponovljenih številk prejetih paketov je prejemnik zmožen zaznavati podvojene sprejeme istih paketov in jih zavreči.

tem tudi dodatno poenostavimo protokol, saj lahko ta vsebuje eno obliko protokolarnega sporočila manj (negativne potrditve niso več potrebne). Prejemnik izvaja posredno potrjevanje tako, da za vsak (pravilno ali nepravilno) prejeti paket odgovori z ACK, ki vključuje številko paketa, ki je *bil nazadnje uspešno sprejet*. V primeru uspešnega sprejema bo torej prejemnik odgovoril z ACK+<številka trenutnega paketa>, v primeru neuspešnega sprejema pa z ACK+<številka prejšnjega paketa>. Če bo prejemnik torej dvakrat prejel potrditev za isti (prejšnji) paket, bo lahko sklepal, da tekoči paket ni bil sprejet pravilno, kar je enakovredno negativni potrditvi.

#### 5.4.5 Tekoče pošiljanje

Protokol, ki smo ga razvili do sedaj, še vedno uporablja sprotno potrjevanje, kar pomeni, da mora čakati na potrditev trenutnega paketa, preden lahko pošlje naslednjega. Tak način pošiljanja bi lahko izboljšali, če bi naredili tak protokol, ki lahko pošlje več paketov hkrati in vodi evidenco, za katere od poslanih paketov je že prejel potrditve. V primeru manjkajočih potrditev bi ta protokol lahko nato poslal ponovno tiste pakete, ki še niso bili potrjeni. Takšen način potrjevanja imenujemo **tekoče pošiljanje ali pošiljanje z drsečim oknom**, omogoča pa nam boljšo izkoriščenost komunikacijskega medija, saj lahko pošiljatelj pošlje naslednji paket hitreje od povratnega časa prenosa med pošiljateljem in prejemnikom.

V nadaljevanju bomo spoznali dva različna protokola za tekoče pošiljanje. Ne glede na principe delovanja vsakega izmed njih, pa ima tekoče pošiljanje nekaj skupnih lastnosti, ki so posledica možnosti hkratnega pošiljanja več paketov skozi omrežje in se odražajo na pošiljatelju in prejemniku:

- Pošiljatelj mora voditi evidenco paketov, v kateri mora razlikovati: pakete, ki so bodo šele prišli na vrsto za pošiljanje (čakajoči paketi); pakete, ki so bili poslani in čakajo na potrditve (nepotrjeni paketi) in pakete, ki so bili poslani ter potrjeni (potrjeni paketi). Poleg paketov pošiljatelj vodi tudi evidenco o velikosti preostalega (prostega) **pomnilnika**, ki ga je namenil hranjenju paketov in ki je običajno v obliki **čakalne vrste**. Vrsta je za ta namen primerna struktura, saj deluje po principu FIFO (angl. *first-in-first-out*), kar sovпадa s tem, da želimo prej poslati tiste pakete, ki so posledica bolj zgodnjih aplikacijskih zahtev. Kadar je najstarejši paket potrjen, se lahko ta iz vrste odstrani, s tem pa se ustvari prostor za novi čakajoči paket.
- Zaradi omejitev pomnilnika in znižanja kompleksnosti protokola je smiselno, da pošiljatelj omeji največje število paketov, ki so v danem trenutku nepotrjeni. Vsakič, kadar prejemnik potrdi nek paket, lahko torej pošiljatelj pošlje enega novega. Ta način delovanja opisujemo s pojmom **drsečega okna**, ki si ga lahko predstavljamo kot čakalno vrsto, iz katere izstopajo potrjeni paketi, vanjo pa vstopajo poslani, a nepotrjeni paketi. Največje

število nepotrjenih paketov, ki so v obtoku, je torej enako parametru, ki ga imenujemo **velikost okna**.

- Ker imamo lahko v obtoku večje število paketov, je potrebno, da **povečamo razpon števil**, ki smo jih uporabljali za številčenje paketov (številki 0 in 1 ne bosta več zadoščali). Le na ta način lahko zagotovimo, da se lahko pošiljatelj in prejemnik nedvoumno sporazumeta, katere številke paketov potrjuje prejemnik.
- Ker je v obtoku lahko večje število paketov, lahko pošiljatelj uporablja tudi **več časovnih kontrol**, za vsak paket največ eno.
- Za lažje potrjevanje bosta pošiljatelj in prejemnik uporabljala posebno obliko posrednega potrjevanja, ki ga imenujemo **kumulativno potrjevanje**. Pri tej obliki potrjevanja bo veljalo, da potrditev oblike  $ACK+<\text{številka paketa } n>$  uspešno potrjuje prejem *vseh preteklih paketov do vključno n-tega*. Na primer, če je pošiljatelj poslal pakete s številkami P35, P36, P37, P38 in P39, prejemnik pa odgovori samo z ACK38, šteje, da je potrdil prve štiri pakete, zadnjega pa (še) ne.
- Ker vemo, da lahko paketi potujejo po različnih poteh, jih bo prejemnik lahko sprejel v **nepravilnem vrstnem redu**. Zato je tudi za prejemnika pomembno, da ima na sprejemni strani namenski pomnilnik, v katerem bo hranil pakete toliko časa, dokler ne pridejo morebitni manjkajoči paketi. Ti paketi so potrebni za to, da prejemnik lahko vse pakete uredi v pravilni vrstni red in uspešno dekapsulira celotno sporočilo.

V nadaljevanju bomo opisali dve obliki protokolov za tekoče pošiljanje, ki uporabljata različne implementacije zgoraj naštetih lastnosti.

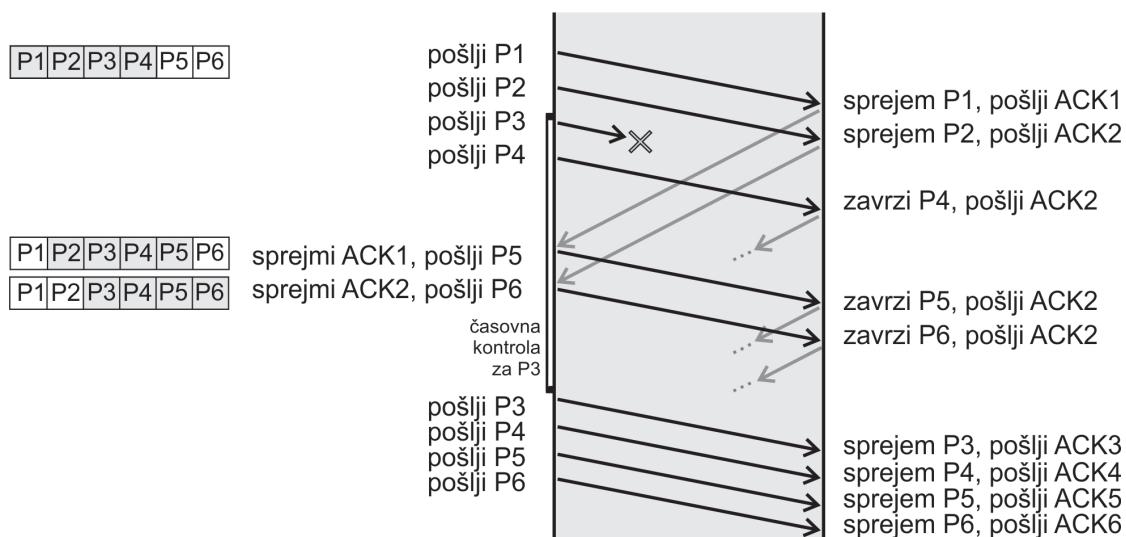
## Ponavljanje zadnjih

Pri ponavljanju zadnjih (angl. *go-back-N*) hrani pošiljatelj okno z največjim številom nepotrjenih paketov, prejemnik pa ne izvaja urejanja paketov, saj protokol zagotavlja sprejem v pravilnem vrstnem redu. Pošiljatelj hrani samo eno časovno kontrolo, ki jo proži za najstarejši paket v oknu. Kadar ta časovna kontrola poteče, pošlje pošiljatelj vse pakete iz okna prejemniku ponovno. Če prejemnik prejme paket, ki ni v pravilnem vrstnem redu, je to znak, da se je vsaj en vmesni paket izgubil. Vsak sprejeti paket, ki ni v pravilnem vrstnem redu, lahko prejemnik zavrže, saj iz zgornjega opisa protokola vemo, da ga bo pošiljatelj ponovno poslal kasneje – v oknu vseh nepotrjenih paketov, v katerem so tudi manjkajoči nepotrjeni paketi.

Za lažje razumevanje delovanja protokola si oglejmo sliko 5.6, ki prikazuje pošiljatelja, ki uporablja velikost okna 4. Pošiljatelj na začetku pošlje štiri pakete (P1, P2, P3 in P4), od katerih prejemnik odgovori s potrditvami ACK1, ACK2,

ACK2 (in ne ACK3!), saj se paket P3 izgubi, P2 pa je bil zadnji še pravilno sprejeti paket. Ker je prejemnik prejel paket P4 v nepričakovanim vrstnem redu (P3 manjka), ga zavrže. V nadaljevanju opazimo, da ko pošiljatelj prejme potrditev ACK1, premakne okno naprej in pošlje naslednji paket v vrsti – P5. Enako se zgodi tudi ob prejeti potrditvi ACK2, ki omogoči pošiljanje paketa P6. V pošiljateljevem oknu so sedaj paketi P3, P4, P5 in P6, časovna kontrola pa meri čas pošiljanja za P3, ki je najstarejši paket v oknu. Prejemnik zavrže tudi P5 in P6, saj tudi ta dva nista v pravilnem vrstnem redu (še vedno manjka P3). Ko poteče časovna kontrola, pošlje pošiljatelj celotno okno paketov ponovno (torej pakete P3, P4, P5 in P6). Prejemnik jih tokrat sprejme v pravilnem vrstnem redu in uspešno potrdi.

Vprašamo se lahko še, kaj bi se zgodilo v primeru izgubljene potrditve. Ker protokol uporablja kumulativno potrjevanje, predstavlja vsaka potrditev paketa z višjo številko hkrati tudi potrditev paketov, za katere se je potrditev izgubila. Protokol je zato odporen proti takšnim scenarijem.



**Slika 5.6** Primer delovanja protokola za tekoče pošiljanje s ponavljanjem zadnjih paketov. Slika prikazuje čakalno vrsto pošiljatelja (na levi) in scenarij ob izgubi paketa P3, ki povzroči pošiljanje celotnega okna paketov.

### Ponavljanje izbranih

Pri ponavljanju izbranih (angl. *selective repeat*) uporablja pošiljatelj okno poslanih paketov, prejemnik pa okno sprejetih paketov. Slednjega mora prejemnik uporabljati zato, ker protokol dopušča sprejem paketov v nepravilnem vrstnem redu. Zato mora prejemnik hraniti pakete do prihoda manjkajočih, da jih lahko

uredi in dostavi aplikaciji. Za prejemnikovo okno velja enako kot za pošiljateljevo: šele kadar je uspešno urejen in sprejet najstarejši paket, lahko prejemnik okno premakne naprej in sprejema nove pakete v čakanje za urejanje. Tak sistem mu omogoča smiselno uporabo pomnilnika, ki je omejen.

Največja razlika od prejšnjega protokola za tekoče pošiljanje je pri ponavljanju izbranih ta, da hrani pošiljatelj časovno kontrolo za vsak posamezni poslani paket posebej (hrani torej toliko časovnih kontrol, kot je velikost okna). Pošiljatelj in prejemnik izvajata potrjevanje posameznih paketov (in ne torej kumulativnega potrjevanja), na podlagi česar lahko pošiljatelj ob poteku časovnih kontrol ponovi pošiljanje samo tistih paketov, za katere še ni prejel potrditve (in ne pošilja vseh paketov v oknu). V primeru, če pride do izgube potrditve, bo pošiljatelj isti paket poslal ponovno, ko poteče časovna kontrola zanj. Prejemnik bo moral zato na podlagi številk paket ugotavljati, ali je prejeti paket že prejel v preteklosti in če gre za duplikat, bo le-tega zavrgel.

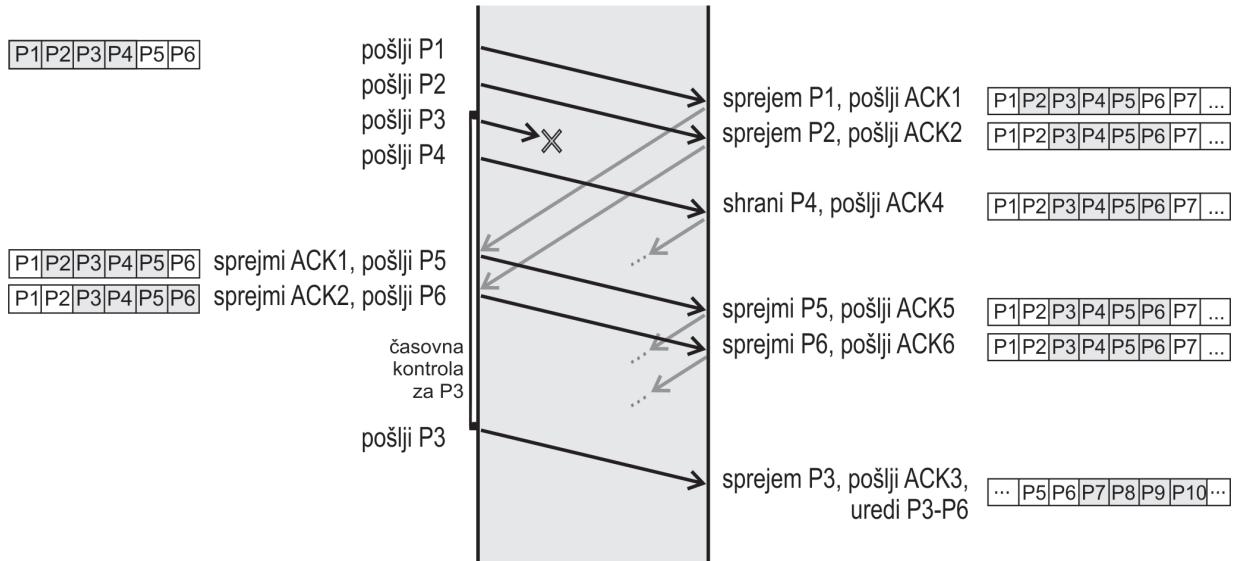
Primer delovanja protokola s ponavljanjem izbranih prikazuje slika 5.7. Slika prikazuje pošiljatelja in prejemnika, ki uporablja okno velikosti 4. Do prve zapolnitve okne pošlje pošiljatelj pakete P1, P2, P3, P4, od katerih se P3 izgubi. Ob pravilnem sprejemu prejemnik potrjuje vsak paket posebej in odgovori s potrditvami ACK1, ACK2 in ACK4. Ker je ob prejemu P4 prejemnik zaznal, da je manjkal paket P3, paketa P4 še ne preda višji plasti, ampak ga začasno shrani. Po prejemu ACK1 in ACK2 premakne pošiljatelj okno naprej in pošlje P5 in P6. Prejemnik tudi tadva potrdi z ACK5 in ACK6 in ju začasno shrani, saj še vedno ni sprejel P3. Ko poteče pošiljateljeva časovna kontrola za najstarejši paket v oknu (P3), pošlje pošiljatelj P3 ponovno. Prejemnik ga tokrat uspešno sprejme, uredi pakete P3, P4, P5 in P6 v pravilni vrstni red in jih dostavi višji plasti. Ob tem se prejemnikovo okno prestavi naprej: pripravljen je na sprejem paketov od P7 naprej.

## 5.5 Protokol TCP

Na podlagi prikazanih principov za zagotavljanje zanesljive dostave lahko končno predstavimo protokol *Transmission Control Protocol* (TCP), ki te principe tudi implementira. Poleg odkrivanja napak, ponovnega pošiljanja, urejanja vrstnega reda in zaznavanja duplikatov skrbi TCP tudi za nadzor pretoka in kontrolo zasičenja. Je povezavni protokol, kar pomeni, da pred prenosom podatkov potrebuje vzpostavitev povezave, pri kateri se določijo potrebni parametri za izvedbo komunikacije. Na povezavi, ki poveže natanko enega pošiljatelja in sprejemnika, lahko teče dvosmerni promet v obliki oštevilčenega toka podatkov. Segment protokola je prikazan na sliki 5.8, njegova polja pa so podrobneje razložena v tabeli 5.2.

| POLJE                    | DOLŽINA              | POMEN                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                    |
|--------------------------|----------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| ŠT. IZVORNIH VRAT        | 16 bitov             | Številka izvornih vrat, ki je običajno najključno določena z vrednostjo $\geq 1024$ in potrebna za odgovor pošiljaljevemu procesu.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                       |
| ŠT. CILJNIH VRAT         | 16 bitov             | Številka, potrebna za naslavljjanje procesa na prejemnikov strani. Za določene (strežniške) aplikacije obstajajo ustaljene (standardizirane) številke vrat.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                              |
| ZAPOREDNA ŠT.            | 32 bitov             | Številka, ki določa zaporedje segmentov, izraženo v številu prenesenih bajtov podatkov.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                  |
| ŠT. POTRDITVE            | 32 bitov             | Številka potrditve, izražena v številki naslednjega pričakovanega bajta s strani prejemnika potrditve.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                   |
| DOLŽINA GLAVE            | 4 biti               | Določa dolžino glave v 32-bitnih besedah.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                |
| REZERVIRANO POLJE        | 3 biti               | Hrani vrednost 0, namenjeno morebitnim razširitvam v prihodnosti.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                        |
| KONTROLNI BITI           | 9 bitov              | <ul style="list-style-type: none"> <li>NS, CWR, ECE: 3 kontrolni biti, ki se v eksperimentalni različici uporabljajo za kontrolo zasičenja z uporabo omrežnih storitev,</li> <li>URG: zastavica, ki indicira veljavno vsebino v polju s kazalcem na nujno vsebino,</li> <li>ACK: zastavica, ki indicira veljavno vsebino v polju s številko potrditve,</li> <li>PSH: zahteva takojšnjo dostavo aplikaciji brez hranjenja v sprejemnem medpomnilniku,</li> <li>RST: zahteva za resetiranje povezave,</li> <li>SYN: zahteva po vzpostavitvi povezave in usklajevanju začetnih zaporednih številk,</li> <li>FIN: zahteva po prekinitvi povezave.</li> </ul> |
| SPREJEMNO OKNO           | 16 bitov             | Uporabno za nadzor pretoka: pošiljaljevo sporočilo o prostem sprejemnem medpomnilniku.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                   |
| KONTROLNA VSOTA          | 16 bitov             | Kontrolna vsota, izračunana z algoritmom <i>Internet Checksum</i> , ki je izračunana nad podatki iz glave paketa IP (naslov pošiljalja, naslov prejemnika, protokol, dolžina) in vsemi podatki iz UDP datagrama.                                                                                                                                                                                                                                                                                                                                                                                                                                         |
| KAZALEC NA NUJNO VSEBINO | 16 bitov             | Odmik glede na zaporedno številko, ki kaže na začetek podatkov, ki naj jih prejemnik preko vrste dostavi aplikaciji. V sodobnem TCP zelo slabo uporabno in implementirano.                                                                                                                                                                                                                                                                                                                                                                                                                                                                               |
| OPCIJE                   | do 320 bitov         | Možne razširitve glave segmenta, dolžina mora biti večkratnik 32 bitov.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                  |
| APL. PODATKI             | spremenljiva dolžina | Enkapsulirani podatki aplikacijske plasti.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                               |

Tabela 5.2 Pomen polj v segmentu protokola TCP



**Slika 5.7** Primer delovanje protokola za tekoče pošiljanje s ponavljanjem izbranih paketov. Slika prikazuje čakalni vrsti pošiljatelja in prejemnika ter scenarij ob izgubi paketa P3. Ob poteku časovnega kontrole se manjkajoči paket pošlje ponovno, prejemnik pa shranjene pakete uredi v pravilni vrstni red.

### 5.5.1 Vzpostavitev povezave in številčenje segmentov

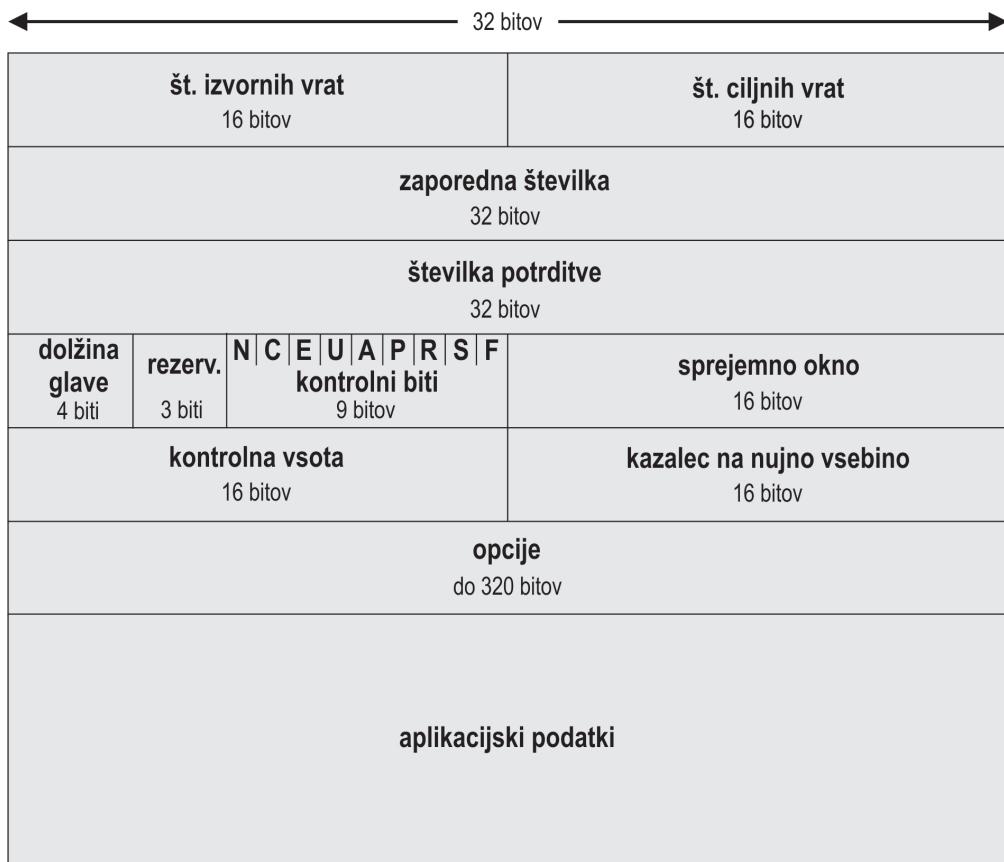
Ker je TCP povezaven protokol, pošiljatelj in prejemnik pred izmenjavo podatkov najprej vzpostavita povezavo. Čeprav povezavo prične vzpostavljati le eden izmed udeležencev, je povezava po vzpostavljivosti dvosmerna, kar pomeni, da se podatki lahko prenašajo v obe smeri. Da bi razlikovali med udeležencem, ki prične z vzpostavljanjem komunikacije od drugega, ki jo sprejme, bomo v nadaljevanju prvega imenovali *pošiljatelj*, drugega pa *prejemnik*, kljub temu da gre za komunikacijsko enakovredna udeleženca.

Vzpostavitev povezave poteka s postopkom, imenovanim **trojno rokovanje** (angl. *three-way handshake*), v katerem udeleženca:

- naključno določita in izmenjata zaporedni številki, s katerima bosta pričela številčiti segmente (začetni številki na obeh straneh sta neodvisni in naključno določeni zaradi večje varnosti – onemogočanja preproste ponovitve komunikacije in vrivanja paketov),
- izmenjata začetne velikosti medpomnilnikov, kar je potrebno za izvajanje nadzora pretoka (glej razdelek 5.5.6).

Trojno rokovanje se izvede v naslednjih treh korakih, ki so prikazani tudi na sliki 5.9:

1. Pošiljatelj pošlje prejemniku segment z zastavico SYN. Ta segment še ne

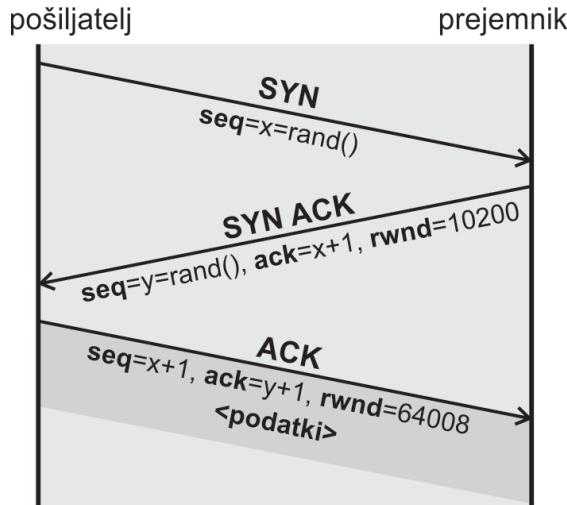


Slika 5.8 Segment protokola TCP

vsebuje podatkov, v polju *zaporedna številka* (na skici označeno s *seq*) pa vsebuje začetno številko zaporedja segmentov, ki je določena naključno.

2. Prejemnik odgovori s segmentom, ki ima veljavni zastavici SYN in ACK. S tem segmentom obvesti pošiljatelja o svoji začetni številki segmenta (v polju *zaporedna številka* oziroma *seq*) in z zastavico ACK in poljem *številka potrditve* (na skici zapisano kot *ack*) potrdi sprejem prejšnjega segmenta. Prejemnik ravno tako obvesti pošiljatelja o svoji velikosti razpoložljivega sprejemnega medpomnilnika (na skici označeno z *rwnd*).
3. Pošiljatelj izvede še tretji korak – prejemniku pošlje segment z zastavico ACK, s katerim potrdi sprejem podatkov iz koraka 2. Pošiljatelj istočasno prejemnika obvesti tudi o trenutni velikosti svojega razpoložljivega medpomnilnika za sprejete segmente. V tem koraku lahko pošiljatelj v segmentu skupaj s potrditvijo ACK že pošilja tudi podatke (skupno pošiljanje potrditve in hkrati novih podatkov imenujemo *pošiljanje na štuporamo* (angl. *piggy-backing*)).

Od tega trenutka dalje je povezava vzpostavljena in udeleženca lahko nadaljujeta z izmenjavo segmentov.



**Slika 5.9** Postopek trojnega rokovanja (seq predstavlja zaporedno številko, ack predstavlja številko potrditve, rwnd pa velikost sprejemnega medpomnilnika)

**Varnostni pomislek:** SYN flood je napad, v katerem napadalec pošlje strežniku veliko število paketov za vzpostavitev povezave (SYN), pri čemer strežnik na zahtevo odgovori s segmentom SYN ACK in za vsako zahtevo rezervira del svojega medpomnilnika. Ker napadalec načrto ne zaključi tretjega koraka trojnega rokovanja (ACK), se zasedenost strežnikovega pomnilnika zaradi delno odprtih povezav povečuje. V skrajni sili lahko pride do prenehanja delovanja strežnika oz. onemogočanja njegovega delovanja (angl. *denial of service, DoS*). Napad, ki ga lahko izvede tudi več napadalcev hkrati (porazdeljeni napad DoS), je potrebno preprečevati z zagotavljanjem dovolj velikega možnega števila povezav, časovno kontrolo hranjenja napol odprtih povezav ali pa s sistemom za opazovanje paketov v omrežju.

Pošiljatelj in prejemnik pri številčenju segmentov ne uporabljata zaporednega številčenja z naravnimi števili, kot smo to simbolično prikazovali v naših učnih scenarijih, ampak uporabljata drugačno definicijo vrednosti v poljih **zaporedna številka** in **številka potrditve**:

- V polje **zaporedna številka** pošiljatelj vedno shrani prvo zaporedno številko bajta, ki tvori sporočilo. Na primer, če pošiljatelj v nekem segmentu, ki ima zaporedno številko 81, pošlje 8 bajtov podatkov (torej pošlje bajte s številkami 81-88), bo njegov naslednji segment imel zaporedno številko 89.
- V polje **številka potrditve** udeleženec v komunikaciji vedno zapiše naslednji zaporedno številko bajta, ki ga pričakuje od nasprotnega udeleženca.

Na primer, če je pošiljatelj v nekem segmentu z zaporedno številko 81 poslal 8 bajtov podatkov (torej bajte s številkami 81-88), bo odjemalec odgovoril s številko potrditve 89, kar je številka naslednjega pričakovanega segmenta s strani pošiljatelja.

Spomimo se, da smo opisani način številčenja paketov in potrditev videli že na sliki 5.9, ki prikazuje pričetek številčenja paketov pošiljatelja začenši z vrednostjo  $x$  in začetek številčenja paketov prejemnika začenši z vrednostjo  $y$ . V drugem in tretjem koraku rokovanja lahko že vidimo, kako prejemnik s številko potrditve zahteva paket s številko  $x + 1$ , tega pa pošiljatelj v naslednjem koraku tudi pošlje.

### 5.5.2 Rušenje povezave

Kadar eden od udeležencev v komunikaciji zaključi s prenosom podatkov, lahko sproži rušenje povezave. Rušenje se izvede v naslednjih štirih korakih:

1. Pošiljatelj pošlje strežniku segment z zastavico FIN.
2. Prejemnik odgovori na prejeti segment s potrditvijo (ACK).
3. Kadar tudi prejemnik zaključi s prenosom podatkov proti pošiljatelju, pošlje tudi prejemnik pošiljatelju segment z zastavico FIN.
4. Pošiljatelj prejemniku odgovori s potrditvijo (ACK) na prejeti zaključni segment. Po pošiljanju pošiljatelj uporabi čakanje s časovnim intervalom, v primeru, če se njegova poslana potrditev izgubi in prejemnik ponovi pošiljanje segmenta FIN. Po preteku časovnega intervala je povezava dokončno zaprta.

### 5.5.3 Stanja protokola TCP

Iz opisanih mehanizmov za vzpostavljanje in rušenje povezave smo lahko razbrali, da se lahko pošiljatelj in odjemalec pri protokolu TCP nahajata v enem izmed več možnih stanj protokola. Sodobni operacijski sistemi nam omogočajo vpogled v seznam trenutnih povezav TCP in v njihovo stanje, ki je lahko eno izmed naslednjih:

- **LISTEN:** stanje prejemnika, v katerem čaka na dohodne zahteve po vzpostaviti povezav,
- **SYN\_SENT:** stanje pošiljatelja, v katerega preide po prvem koraku trojnega rokovanja (SYN),
- **SYN\_RECV:** stanje prejemnika, v katerega preide po izvedenem drugem koraku trojnega rokovanja (SYN ACK),
- **ESTABLISHED:** stanje pošiljatelja, v katerega preide po izvedenem tretjem koraku trojnega rokovanja (ACK) in stanje prejemnika, v katerega

le-ta preide ob prejemu istega ACK; v tem stanju je povezava uspešno vzpostavljena in udeleženca lahko izmenjujeta segmente,

- **FIN\_WAIT\_1**: stanje, v katerega preide pošiljatelj ob izvedenem prvem koraku rušenja povezave (FIN),
- **CLOSE\_WAIT**: stanje, v katerega preide prejemnik po sprejemu segmenta FIN in odgovoru z ACK,
- **FIN\_WAIT\_2**: stanje, v katerega preide pošiljatelj ob sprejemu ACK prejemnika in čakanju na njegov FIN,
- **LAST\_ACK**: stanje, v katerega preide prejemnik ob izvedbi tretjega koraka rušenja povezave (pošiljanje FIN),
- **TIME\_WAIT**: stanje, v katerega preide pošiljatelj ob prejemu prejemnikovega FIN in odgovorom z ACK; v tem stanju pošiljatelj izvaja čakanje za morebitno prestrezanje zamujenih zaključnih segmentov<sup>3</sup>,
- **CLOSED**: stanje, v katerega preide pošiljatelj in prejemnik po uspešnem rušenju povezave.

#### 5.5.4 Nastavitev časovne kontrole

Spoznali smo, da se za zagotavljanje zanesljive dostave segmenta uporablja časovna kontrola, znotraj katere pošiljatelj pričakuje potrditev s strani prejemnika. V tem razdelku bomo opisali, kako protokol TCP nastavlja dolžino časovne kontrole.

Pri nastavljanju dolžine časovne kontrole se pošiljatelj opira na meritve povratnega časa komunikacije (angl. *round-trip time, RTT*) do prejemnika. To je čas, ki je potreben za to, da pošiljatelj pošlje prejemniku okvir, prejemnik pa da nanj odgovori. Intuitivno se lahko zavedamo, da dolžina časovne kontrole ne sme biti manjša od tega najkrajšega časa komunikacije, saj bo pošiljatelj v tem primeru "neučakano" oz. prezgodaj prožil ponovno pošiljanje segmentov. Ravno tako pa nastavitev časovne kontrole ne sme biti veliko daljša od časa RTT, saj v primeru izgubljenih paketov to predstavlja predolgo čakanje, ki vodi v slabo izkoriščenost komunikacijskega kanala.

Ker se dolžina RTT lahko zaradi različnih poti, obremenjenosti usmerjevalnikov in zasičenosti omrežja skozi čas spreminja, mora tudi pošiljatelj biti sposoben dolžino časovnega kontrole dinamično prilagajati razmeram. Za izračun primerne časovne kontrole zato pošiljatelj avtomatsko opravlja meritve RTT skozi čas. Na podlagi izmerjenih RTT (količina  $IzmerjeniRTT$ ) gladi skoke v meritvah z izračunom gibajočega povprečja skozi čas (količina  $OcenjeniRTT$ ), pri katerem s faktorjem  $\alpha$  uteži novo vrednost izmerjenega RTT, s faktorjem  $1 - \alpha$  pa uteži vrednost gibajočega povprečja v prejšnji časovni enoti. Vrednost gibajočega

<sup>3</sup>Glede na RFC793 je najdaljši čas čakanja definiran s dvakratnikom dolžine parametra MSL (*Maximum Segment Lifetime*) in znaša 4 minute.

povprečja v novi časovni enoti izračuna kot linearno kombinacijo obeh količin, pri čemer običajno uporabi  $\alpha = 0,125$ :

$$\begin{aligned} OcenjeniRTT[i] &\leftarrow (1 - \alpha) \cdot OcenjeniRTT[i - 1] \\ &\quad + \alpha \cdot IzmerjeniRTT[i] \end{aligned}$$

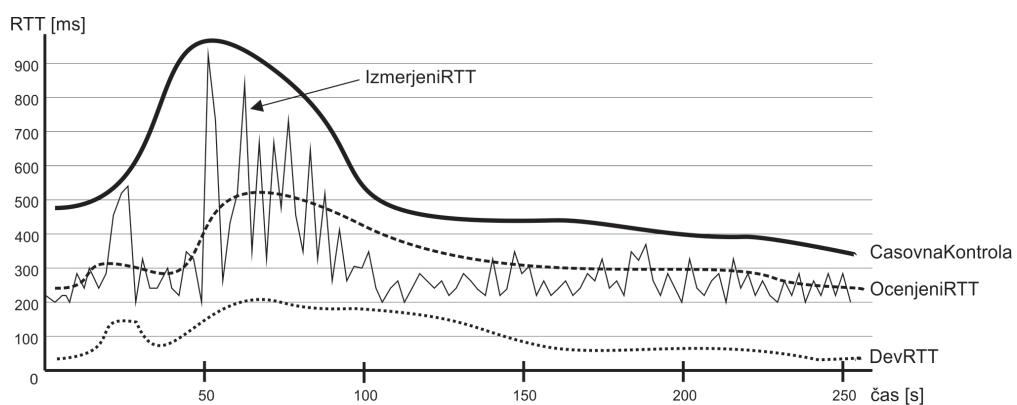
Nadalje pošiljatelj izračuna tudi gibajočo deviacijo, ki je količina, ki opisuje zglajena odstopanja od gibajočega povprečja. Tudi njo izrazi kot linearno kombinacijo vrednosti deviacije v prejšnji časovni enoti, utežene z  $\beta$ , in vrednosti absolutne razlike med izmerjenim RTT in gibajočim povprečjem, utežene z  $1 - \beta$  (privzeta vrednost za  $\beta$  znaša 0,25):

$$\begin{aligned} DevRTT[i] &\leftarrow (1 - \beta) \cdot DevRTT[i - 1] \\ &\quad + \beta \cdot |IzmerjeniRTT[i] - OcenjeniRTT[i]| \end{aligned}$$

Izračunano gibajočo deviacijo, ki opisuje velikost odstopanj od povprečja, lahko pošiljatelj uporabi kot "rezervo", ki jo mora prištetи gibajočemu povprečju, da skupna vsota zagotovo ne predstavlja prekratkih časovnih kontrol. Implementacije protokola TCP običajno uporabljajo štirikratnik gibajoče deviacije kot privzeto "rezervo" in dolžino časovne kontrole izrazijo kot:

$$CasovnaKontrola[i] = OcenjeniRTT[i] + 4 * DevRTT[i].$$

Vizualni primer računanja časovne kontrole na podlagi izmerjenih časov RTT je prikazan na sliki 5.10.



**Slika 5.10** Izračun dolžine časovne kontrole na podlagi gibajočega povprečja in gibajoče deviacije

### 5.5.5 Potrjevanje TCP

V razdelkih 5.4.5 in 5.4.5 smo spoznali dva pristopa k implementaciji tekočega pošiljanja s potrjevanjem. Protokol TCP uporablja kombinacijo teh dveh pristopov: bolj je podoben pošiljanju zadnjih (angl. *go-back-N*) s to razliko, da ob poteku časovne kontrole ponovi le pošiljanje najstarejšega segmenta v oknu in ne celotnega okna segmentov<sup>4</sup>.

Spomnimo se lahko, da je prejemnik pri protokolu za pošiljanje zadnjih segmentov na račun ponovnega pošiljanja celotnega okna lahko zavračal segmente, ki so prispeli v nepravilnem vrstnem redu. Ker pa pri protokolu TCP pošiljatelj ne pošilja več celotnega okna, mora TCP prejemnik hrani nepravilno sprejete segmente v medpomnilniku in jih urediti v pravilni vrstni red ob prispetju manjkajočih. Vidimo lahko, da je ta del protokola bolj podoben protokolu za ponavljanje izbranih segmentov. V primeru, da se poslani segment izgubi in da prejemnik prejme segment s previsoko številko, prejemnik **zazna vrzel** v številkah. V tem primeru prejemnik odgovori s številko potrditve tistega segmenta, ki je še zadnji prišel v pravilnem vrstnem redu. Kadar pošiljatelj ponovi pošiljanje manjkajočih segmentov, začne s tem prejemniku **polniti vrzel** v manjkajočih številkah. V tem primeru prejemnik sproti potrjuje pakete, ki so mu do tega trenutka manjkali.

Poleg opisanih splošnih principov potrjevanja ima TCP še nekaj specifik:

- Če je prejemnik vse prejšnje segmente že potrdil, potem ob sprejemu naslednjega segmenta v pravilnem vrstnem redu prejemnik izvede **zakasnjeno potrjevanje** (angl. *delayed acknowledgment*). Pri zakasnjenem potrjevanju prejemnik ne odgovori s potrditvijo takoj, temveč počaka 500 ms. V kolikor v tem času pride še drugi novi segment, prejemnik izvede kumulativno potrditev obeh tako, da potrdi le zadnji prejeti segment, s čimer implicitno potrjuje uspešen prejem tudi prvega. Poudarimo, da po prejemu drugega segmenta prejemnik ne izvaja ponovnega čakanja intervala 500 ms, temveč mora nujno poslati potrditev. V kolikor v intervalu 500 ms prejemnik ne sprejme še drugega segmenta, potrdi samo prvega po preteklu intervala. Opisani postopek omogoča, da lahko prejemnik pošilja potrditve samo za vsaki drugi segment, s čimer se zmanjša potrebna količina režije pri pošiljanju. Poudarimo še, da prejemnik ob polnjenju vrzeli ne uporablja zakasnjenega potrjevanja, ampak brez zakasnitve potrdi vsak prej manjkajoči paket.
- V primeru, da pošiljatelj pošlje pet paketov (P1, P2, P3, P4 in P5) in od prejemnika prejme potrditev za prvi paket (ACK1), ki ji sledijo še tri kopije iste potrditve (ACK1, ACK1, ACK1; skupno torej štiri enake potrditve) je to znamenje, da je prejemnik uspešno prejel P1, P3, P4 in P5, paket P2 pa da se je izgubil (tri kasnejše enake potrditve so posledica zaznane

---

<sup>4</sup>V zgodovini so obstajali tudi predlogi potrjevanja izbranih paketov v oknu (glej RFC2018)

vrzeli pri prejemu segmentov). Ob takšnem scenariju pošiljatelj izvede postopek **hitrega ponovnega pošiljanja** (angl. *fast retransmit*). Pri tem postopku pošiljatelj ne čaka na potek časovne kontrole za potrditev paketa P2, temveč takoj ob prejemu četrte iste potrditve ponovi pošiljanje P2. S tem postopkom pošiljatelj skrajša nepotreben čas za ponovno pošiljanje in prepreči preveliko kopičenje paketov v prejemnikovem sprejemnem medpomnilniku.

### 5.5.6 Nadzor pretoka

Nadzor pretoka je mehanizem, s katerim pošiljatelj in prejemnik usklajujeta hitrost komuniciranja. Pri pošiljanju mora pošiljatelj namreč paziti, da prejemniku ne pošlje prevelike količine podatkov, s čimer lahko prekorači velikost prejemnikovega medpomnilnika. V primeru prekoračitve prejemnik ne more sprejeti poslanih segmentov, kar pomeni, da jih bo potrebno poslati ponovno, s tem pa je izraba komunikacijskega medija manj učinkovita.

Z načinom številčenja segmentov TCP, ki temelji na številkah prvega bajta v segmentu, lahko prejemnik spremlja, koliko podatkov ima v sprejemnem medpomnilniku. Količina vsebovanih podatkov je namreč enaka razliki med številko zadnjega sprejetega bajta (številka zadnjega segmenta + njegova dolžina) in številko zadnjega bajta, ki je bil predan aplikaciji in s tem odstranjen iz medpomnilnika. Če prejemnik od skupne velikosti svojega medpomnilnika odšteje zgornjo količino vsebovanih podatkov, izračuna velikost razpoložljivega prostora oziroma **velikost sprejemnega okna** (angl. *receive window*), ki jo označimo z **rwnd**.

Iz strukture segmenta TCP (slika 5.8 in tabela 5.2) smo videli, da segment vsebuje 16-bitno polje, imenovano *sprejemno okno*. Natanko to polje je tisto, v katerega pošiljatelj in prejemnik pri vsakem pošiljanju zapišeta svojo vrednost **rwnd**, s čimer z vsakim poslanim paketom obveščata sogovornika o velikosti svojega razpoložljivega medpomnilnika v danem trenutku.

Sedaj, ko poznamo definicijo sprejemnega okna in podrobnosti glede njegove uporabe, lahko bolje razumemo tudi, zakaj morata udeleženca ob vzpostaviti povezave (trojno rokovanje) izmenjati tudi podatke o začetnih velikostih njunih medpomnilnikov, kot smo to navedli v razdelku 5.5.1. Sedaj lahko tudi v celoti razumemo vsebino ilustracije rokovanja na sliki 5.9, ki od drugega koraka dalje že prikazuje izmenjavo podatkov o velikosti sprejemnih oken v polju **rwnd**.

### 5.5.7 Nadzor zasičenja

Zasičenje je stanje omrežja, ko zaradi preobilice omrežnega prometa omrežje ne more več opravljati svoje funkcije. Do tega pojava pripelje množica dejavnikov, kot so hitrost procesiranja pošiljateljev in prejemnikov, hitrost delovanja usmerje-

valnikov, velikosti medpomnilnikov in zaporedna narava povezanosti pošiljateljev in prejemnikov. Ker so čakalne vrste (medpomnilnik) v usmerjevalnikih končno velike, usmerjevalniki pa obdelujejo pakete z omejeno hitrostjo, lahko prehitro pošiljanje s strani pošiljateljev povzroči bodisi predolgo polnjenje čakalnih vrst (kadar je prostora še dovolj) in s tem velike zakasnitve ali pa izgubo segmentov (ob prekoračitvi velikosti pomnilnika). Prevelike zakasnitve lahko nadalje povzročijo potek časovnih kontrol zaradi pakete, kar ima isti učinek kot izguba segmentov – pošiljatelj ponovi pošiljanje paketa. Posledica izgubljenih paketov in ponovno poslanih duplikatov sproži začaran krog – še več omrežnega prometa – zaradi česar začne omrežje izvajati večjo količino prenosa, a z veliko manjšim učinkom. Dodaten vidik težave povzroča še to, da če se paket na enem od več potrebnih skokov izgubi, smo s tem izgubili vložene omrežne resurse za vse pretekle skoke. Zaporedje opisanih dogodkov lahko vodi do popolnega **kolapsa omrežja** ob zasičenju (angl. *congestion collapse*).

Za preprečevanje zasičenja omrežja prevladujeta dve družini tehnik:

- **Pristopi z odprto zanko:** izvajata jih samostojno bodisi pošiljatelj ali prejemnik. Vključujejo lahko različne mehanizme za odločanje o smiselnosti ponovnega pošiljanja paketov, velikosti pošiljateljevega okna, zavrnjenje paketov (npr. pri multimedijiških aplikacijah), politiko potrjevanja (uporabo kumulativnega namesto posameznega), možnostih za vzpostavljanje dodatnih novih povezav (pri omrežjih z navideznimi vodi). Takšno vrsto pristopov večinoma uporablja tudi protokol TCP, ki na strani pošiljatelja ob opazovanju prejetih potrditev prilagaja hitrost pošiljanja.
- **Pristopi z zaprto zanko:** izvajata jih pošiljatelj in prejemnik (ali usmerjevalniki v omrežju) v medsebojnem sodelovanju. Vključujejo lahko mehanizme, ko sprejemna naprava zavrača dohodni promet, ki bi lahko povečal zasičenje, ali pa pošiljatelja s posebnimi sporočili obvesti, da omrežje prehaja v stanje zasičenja. O zasičenju je lahko obveščen tudi prejemnik, ki temu ustrezno prilagodi svojo politiko pošiljanja podatkov in potrditev. Primer sporočil za obveščanje o zasičenju so sporočila ECN (angl. *explicit congestion notification*), ki so bila eksperimentalno vključena tudi v protokola IP in TCP<sup>5</sup>, vendar sta manj v uporabi.

Protokol TCP pristopa k nadzoru zasičenja tako, da na strani pošiljatelja prilagaja hitrost pošiljanja. Želja pošiljatelja je, da pošilja svoje segmente čim hitreje, vendar še pod mejo zasičenja omrežja. Na podaljševanje čakalnih vrst v usmerjevalnikih in s tem zakasnitev v omrežju pošiljatelj sklepa z opazovanjem prejetih potrditev za poslane pakete. V kolikor za poslani paket znotraj predvidene časovne kontrole prejme potrditev, šteje pošiljatelj to za pozitiven dogodek, na podlagi katerega lahko še dodatno poveča hitrost pošiljanja. V nasprotnem

---

<sup>5</sup>RFC3168

primeru, če potrditve ne prejme, pošiljatelj sklepa, da omrežje prehaja v zasičenje in posledično zmanjša hitrost. Če predstavimo čas na vodoravni osi grafa, hitrost pošiljanja pa na navpični, ima torej protokol TCP žagasto obliko funkcije hitrosti. To obliko tvori zaporedje postopnih povečevanj in naglih znižanj hitrosti pošiljanja.

Hitost pošiljanja pošiljatelj nadzoruje z velikostjo **okna za nadzor zasičenja** (angl. *congestion window*), ki ga označimo s **cwnd**. Obe okni (okno za nadzor zasičenja **cwnd** in okno za nadzor pretoka **rwnd**) predstavljata omejitvi hitrosti pošiljanja, zato je samoumevno, da pošiljatelj ne sme pošiljati hitreje kot  $\min(cwnd, rwnd)$ , saj bo v nasprotnem primeru bodisi preobremenil prejemnika ali pa omrežje.

Velikost okna **cwnd** merimo v enotah **MSS** (angl. *maximum segment size*), ki opredeljujejo velikost največjega dovoljega segmenta TCP (brez glave). V Internetu je enota **MSS** neposredno povezana z enoto **MTU** (angl. *maximum transmission unit*), ki določa največjo dolžino okvirja na povezavni plasti. Vemo namreč, da so v datagram povezavne plasti enkapsulirani paketi IP, vanje pa segmenti TCP. Največja dovoljena velikost segmenta **MSS** je torej manjša le za razliko velikosti glav protokolov IP in TCP, od katerih dolžina vsake privzeto znaša 20 bajtov. To pomeni, da privzeto velja  $MSS = MTU - 40 B$ .

Omenimo, da se protokol TCP razvija že od sedemdesetih let in da je skozi zgodovino obstajalo veliko število njegovih različic, od katerih ima vsaka svoje specifike nadzorovanja zasičenja. V nadaljevanju bomo opisali različico TCP Reno, ki je nastala v devetdesetih letih in je zasnovana na konceptih, na katerih temeljijo tudi nadaljnje različice. Nadzorovanje zasičenja v različici TCP Reno ima naslednje tri faze:

1. **Počasni začetek** (angl. *slow start*): Začetna faza, v kateri pošiljatelj prične z velikostjo okna  $cwnd = 1 \text{ MSS}$ . V tej fazi pošiljatelj za vsako uspešno prejeto potrditev poslanega paketa poveča okno **cwnd** za 1 (torej  $cwnd \leftarrow cwnd + 1$ ). Čeprav ima pravilo za povečevanje izgled linearne funkcije, poteka pospeševanje hitrosti pravzaprav eksponentno: prične se s  $cwnd = 1 \text{ MSS}$ , nato se poveča na  $cwnd = 2 \text{ MSS}$ , nato na  $cwnd = 4 \text{ MSS}$  (za vsako potrditev dveh poslanih paketov v prejšnjem koraku se **cwnd** poveča za 1; ker sta bila poslana 2 paketa, se torej **cwnd** z 2 poveča na 4), nato na  $cwnd = 8 \text{ MSS}$  itd. Pošiljatelj povečuje hitrost na opisani način, vse dokler ne pride do izgube paketa. Takrat si zapomni polovico trenutne vrednosti **cwnd** (torej prejšnjo vrednost **cwnd**, pri kateri izgube še ni bilo) kot spremenljivko **prag** (angl. *threshold*) in zniža hitrost pošiljanja na  $cwnd = 1 \text{ MSS}$  ter začne s fazo počasnega začetka znova. V drugi iteraciji povečevanja hitrosti postopa pošiljatelj nekoliko drugače, saj ima tokrat nastavljeno vrednost praga, ki je v prvi iteraciji ni imel. Kadar pri povečevanju hitrosti doseže (ali preseže) velikost praga, pošiljatelj zaključi s fazo počasnega začetka in preide v drugo fazo, fazo *izogibanja zasičenju*.

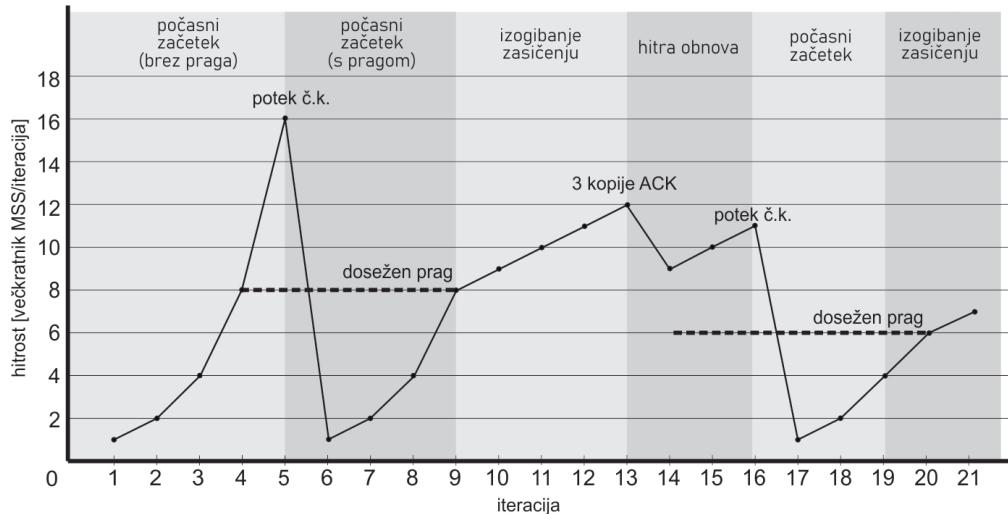
2. **Izogibanje zasičenju** (angl. *congestion avoidance*): Po tem, ko pošiljatelj s svojo hitrostjo v fazi počasnega začetka doseže (ali preseže) vrednost praga, začne hitrost povečevati linearno in ne več eksponentno. To izvede tako, da za celotno množico prejetih potrditev, zahtevanih s trenutno vrednostjo  $cwnd$ , poveča velikost okna za 1 (torej  $cwnd \leftarrow cwnd + 1$ ). To pomeni, da bo npr. pri  $cwnd = 16 \text{ MSS}$  šele po prejetih 16 potrditvah (in ne za vsako posebej kot pri počasnem začetku) povečal velikost okna na  $cwnd = 17 \text{ MSS}$  itd. Na ta način se lahko pošiljatelj bolj počasi približa pragu zasičenja, ki ga je v prvi iteraciji faze počasnega začetka prekoračil. Kadar pride do izgube paketa (manjkajoče potrditve), lahko pošiljatelj ponovno sklepa, da omrežje prehaja v stanje zasičenja in zmanjša hitrost. Pri različici TCP Tahoe (1988), ki je starejša od TCP Reno, je pošiljatelj v tem primeru znižal hitrost na  $cwnd = 1 \text{ MSS}$ , pri različici TCP Reno pa preide v fazo *hitre obnove*.
3. **Hitra obnova** (angl. *fast recovery*): V fazo hitre obnove preide pošiljatelj v primeru izgube paketa, ki jo zazna tako, da prejme tri kopije iste potrditve. V razdelku 5.5.5 smo omenili, da pošiljatelj v primeru prejema treh kopij iste potrditve ne čaka na potek časovnega intervala, temveč izvede hitro ponovno pošiljanje (angl. *fast retransmit*). V postopku nadzorovanja zasičenja pošiljatelj obenem zniža velikost praga na polovico trenutne vrednosti in nastavi  $cwnd \leftarrow cwnd/2 + 3$ , kjer s prišteto vrednostjo 3 upošteva, da so podvojene potrditve uspešno potrdile 3 pakete, ki sledijo izgubljenemu. Kadar v fazi hitre obnove prejemnik uspešno potrdi novi paket (s čimer kumulativno potrije tudi izgubljenega), se pošiljatelj vrne nazaj v fazo izogibanja zasičenju. S fazo hitre obnove torej TCP Reno bolje optimizira hitrost pošiljanja in v primeru izgube paketa v manjšem obsegu zniža hitrost pošiljanja, dokler izgubljeni paket ni potrjen.

Primer možnega scenarija delovanja vseh treh faz nadzora zasičenja prikazuje slika 5.11.

Kot smo že omenili, različne različice protokola TCP so zasičenje nadzorovale z različnimi mehanizmi. Omenimo na tem mestu različico TCP Vegas, ki poleg dodatnih načinov za zaznavanje izgube paketov uvaja tudi linearno zmanjševanje hitrosti pošiljanja v primeru izgub.

## 5.6 Pravičnost TCP in UDP

Ob zaključku poglavja o transportnih protokolih omrežja Internet lahko pozornost usmerimo še na lastnosti sobivanja obeh protokolov TCP in UDP pri njuni sočasni uporabi. Kljub temu, da se primernost tih dveh protokolov razlikuje glede na različne možne aplikacije, si želimo, da vsak od njiju omogoča *pravično* rabo prenosne kapacitete. Kriterij *pravičnosti* pomeni, da želimo vsakemu od  $N$



**Slika 5.11** Primer delovanja vseh treh faz sistema za nadzor zasičenja TCP

uporabnikov, ne glede na izbrani transportni protokol, zagotoviti enakomeren delež celotne kapacitete  $C$ , torej tako, da ima vsak uporabnik razpoložljivih količino  $C/N$  prenosne kapacitete.

### 5.6.1 Pravičnost TCP

Oglejmo si najprej, kako je s pravičnostjo, če imamo samo uporabnike, ki uporabljajo protokol TCP. Izkaže se, da v tem primeru mehanizem za nadzor zasičenja samodejno poskrbi, da se sčasoma uporabljana kapaciteta prenosa med uporabniki izenači in s tem konvergira v pravično delitev.

Da lažje pojasnimo, zakaj sistem več uporabnikov konvergira v pravično delitev kapacitete, si zamislimo omejen scenarij uporabe protokola TCP, ki je ilustriran tudi na sliki 5.12. Denimo, da v tem scenariju nastopata dva uporabnika protokola TCP, ki si delita skupno povezano kapacitete  $C$ . Vsaka od osi v tem grafu prikazuje hitrost, s katero pošilja vsak od uporabnikov, vsaka točka (koordinata) znotraj grafa pa prikazuje različno istočasno kombinacijo hitrosti prvega in drugega uporabnika.

Najprej se lahko vprašamo, kje se na grafu odraža najvišja razpoložljiva prenosna kapaciteta. To lahko ilustriramo s padajočo premico, ki povezuje vrednosti  $C$  na obeh oseh. Ta padajoča premica prikazuje, kako si uporabniki lahko razdelita skupno kapaciteto, ki je izkoriščena v celoti. Vidimo lahko, da premica ponazarja različne scenarije te delitve: točki  $(C, 0)$  in  $(0, C)$ , kjer premica seka osi, prikazujeta scenarija, kjer eden od uporabnikov zaseže celotno kapaciteto, drugi pa ima ničelno. Medtem, ko vmesne točke med tem skrajnima scenarijema prikazujejo različne načine delitve, nas najbolj zanima pravična delitev –

to pa je ravno sredinska točka s koordinato  $(\frac{C}{2}, \frac{C}{2})$ . Poleg točke  $(\frac{C}{2}, \frac{C}{2})$ , kjer je izkoriščena celotna možna prenosna kapaciteta, pa pravično delitev predstavlja tudi vse druge točke, ki so na diagonali kvadranta: to so npr. točke  $(\frac{C}{i}, \frac{C}{i})$ , kjer je  $i \geq 1$ . Padajoča premica torej v kvadrantu razmejuje točke, ki ponazarjajo možne razdelitve kapacitete (pod diagonalo) in nemogoče (nad diagonalo), ker presegajo skupno kapaciteto.

Predpostavimo, da v našem poenostavljenem scenariju velja pravilo, da ob vsaki uspešno prejeti potrditvi pošiljatelj hitrost linearno povečuje, ob izgubi pa prepolovi. Zavedamo se, da takšen način nadzora zasičenja ne izvaja nobena od prej predstavljenih različic protokola TCP, vendarle pa tak umeten scenarij zadošča za opazovanje lastnosti protokola glede delitve hitrosti med oba uporabnika. Naše ugotovitve bomo kasneje tudi zlahka kasneje preslikali v realni svet in na situacije z več uporabniki.

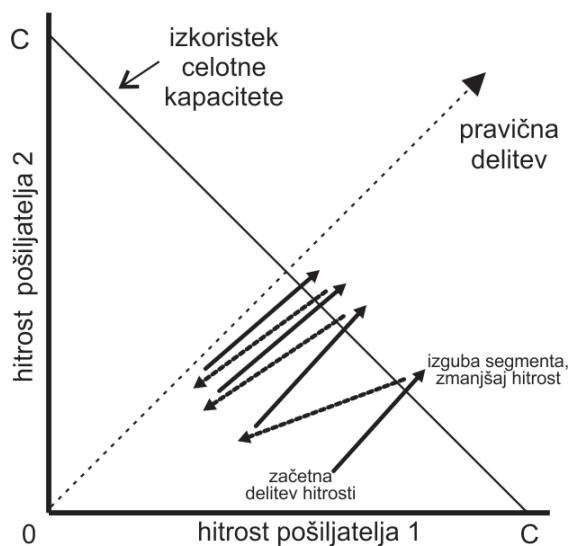
Denimo, da pošiljatelja v nekem trenutku začneta oddajati vsak s svojo hitrostjo  $(C_1, C_2)$ , pri čemer je skupna hitrost manjša od kapacitete kanala. Ob prejemanju uspešnih potrditev poslanih segmentov bosta vsak zase svojo hitrost linearno povečevala (na grafu to vidimo kot premik iz začetne točke v točko v smeri desno in gor, prikazano s puščico). Ko uporabnika v nekem trenutku toliko povečata svoji hitrosti toliko, da presežeta skupno kapaciteto (in s tem preideta nad diagonalo), jima izguba segmenta (manjkajoča potrditev) pove, da morata zmanjšati svoji hitrosti, kar ustreza prepolavljanju vrednosti koordinat točke na grafu (prikazano s črtkano puščico). S tem si pošiljatelja nastavita hitrosti pošiljanja, ki sta bližje diagonali kvadranta in s tem pravični delitvi. Vidimo, da se postopek iterativno nadaljuje na enak način, rezultat vsake iteracije povečevanja in zmanjševanja hitrosti pa je, da se vse bolj približujemo sredinski točki grafa, ki predstavlja optimalno pravično delitev.

Zaključimo lahko, da je zasnova protokola TCP že sama po sebi taka, da nadzor zasičenja samodejno skrbi za pravično delitev. Opisani scenarij zlahka posplošimo tudi ko je prisotnih več pošiljateljev.

### 5.6.2 Pravičnost UDP

Ker protokol UDP ne izvaja nadzora zamašitve, lahko pri njem pričakujemo popolnoma drugačno situacijo glede pravičnosti. Ker UDP ne nadzoruje hitrosti pošiljanja in ne skrbi za zanesljivo dostavo, so uporabniki protokola UDP prepuščeni borbi za prosto kapaciteto, izgubljeni datagrami pa so skrb aplikacije.

V primeru, kadar v istem omrežju sobivajo uporabniki protokolov UDP in TCP, bo pri delitvi kapacitete vedno trpel TCP, saj bo pri izgubah omejeval svojo hitrost, medtem ko bo UDP pošiljal brez omejitev. Pogosto se lahko zgodi, da je zaradi preobsežnega prometa UDP promet s protokolom TCP popolnoma



**Slika 5.12** Spreminjanje hitrosti dveh pošiljateljev v smeri pravične delitve skupne kapacitete C

neuspešen. Snovalci Interneta zato pogosto svarijo in izdajajo priporočila<sup>6</sup> za snovanje aplikacij, ki uporabljajo protokol UDP, tako da se te ustrezeno ozirajo tudi na druge sočasne prenose in se poskušajo izogniti zasičenju.

<sup>6</sup><https://tools.ietf.org/id/draft-ietf-tsvwg-udp-guidelines-04.html>

## 6 Aplikacijska plast

Če se spomnimo *storitvene definicije* omrežja, ki smo jo spoznali v uvodu, smo omrežje definirali kot *sredstvo*, ki nam omogoča izvajati omrežne storitve. Ravno zaradi tega je aplikacijska plast ključnega pomena, saj aplikacije, ki jih želimo izvajati v omrežju, za svojo komunikacijo uporabljajo t.i. aplikacijske protokole. Iz tega, da je aplikacijska plast postavljena na vrh komunikacijskega sklada, sledi, da so aplikacijski protokoli tisti, ki odjemajo storitve nižje (transportne) plasti, kar se nato nadaljuje navzdol po komunikacijskem skladu, kot smo že spoznali. Ker so storitve aplikacijske plasti ključ za obstoj omrežij, se nanje v literaturi velikokrat sklicujejo s francosko besedo zvezo *raison d'être* (angl. *reason to be*, slov. *razlog za obstoj*).

Ker besedo *aplikacija* poznamo že iz vsakodnevne rabe, je na tem mestu vredno poudariti razliko med pomenom (navadne) aplikacije in *omrežne ali porazdeljene* aplikacije. Najbolj pogosta raba besede *aplikacija* je namenjena sklicevanju na del programske opreme, ki se izvaja znotraj računalniškega sistema, običajno znotraj operacijskega sistema. Primeri takšnih vsakodnevnih aplikacij so urejevalniki besedil, programi za delo s preglednicami, programski jeziki, matematična orodja, predvajalniki videa in zvoka ipd. Za razliko od teh aplikacij se *omrežne ali porazdeljene* aplikacije izvajajo na več računalniških sistemih hkrati, od koder tudi izvira njihova oznaka, da so *porazdeljene*. Te aplikacije so torej sestavljene iz več programskih komponent, od katerih se vsaka izvaja na drugem računalniškem sistemu. Za delovanje celotne aplikacije je potrebno sodelovanje vseh komponent, saj storitev sicer ne deluje. Primer takšnih porazdeljenih aplikacij so npr. številni sistemi, ki delujejo v arhitekturi odjemalec-strežnik ali pa protokoli, ki tečejo med končnimi sistemi (P2P, angl. *peer-to-peer*). Kot primere porazdeljenih aplikacij lahko navedemo: e-mail, splet (angl. *world wide web*), takojšnje sporočanje, oddaljen dostop, izmenjava datotek, omrežne igre, multimedijijske aplikacije, telefonija IP in konferenčne aplikacije, sisteme za porazdeljeno procesiranje, pa tudi sistem za samodejno nastavitev DHCP, imenska storitev DNS in številne druge.

S seznama porazdeljenih aplikacij lahko razberemo, da njihove komponente ne potrebujejo nujno grafičnega uporabniškega vmesnika (npr. spletni strežnik, e-mail strežnik, telefonija IP), in da je posamezne komponente omrežnih aplikacij

za medsebojno komunikacijo uporabljajo **aplikacijske protokole**. Naloga programerja, ki želi razviti omrežno aplikacijo, je torej svoje delo pretežno posvetiti razvoju programske kode aplikacije in uporabi (ali tudi razvoju) aplikacijskega protokola, ki ga bo aplikacija uporabljala. Pri tem pa lahko podrobnosti omrežja zanemari, saj lahko v razviti programski kodi le preprosto zahteva izvedbo storitev prenosa aplikacijskih sporočil, ki transparentno tečejo na nižjih plasteh komunikacijskega modela. Zaradi omejitev velikosti datagramov nižjih plasti (spomnimo se na MTU in MSS), se daljsa aplikacijska sporočila razdelijo v več segmentov transportne plasti.

V svetu obstajajo že številni aplikacijski protokoli, ki so uveljavljen standard za najbolj razširjene omrežne aplikacije. V grobem jih lahko razdelimo v dve skupini: javne (odprte) in lastniške (zaprte, angl. *proprietary*). Javni standardizirani protokoli, kot so npr. BitTorrent, DNS, HTTP, IMAP, LDAP, NTP, POP3, RDP, RTP, SMTP in SSH, so do potankosti definirani v javnodostopnih dokumentih, kot so običajno specifikacije RFC (angl. *Request For Comments*)<sup>1</sup>, ki so objavljeni na straneh organizacije IETF. Poleg uveljavljenega standarda predstavlja naziv dokumentov RFC tudi poziv skupnosti k podajanju pripomb in izboljšav na opisane zaslove protokolov. Pri pregledovanju delovanja pomembnejših protokolov Interneta bomo videli, da je veliko aplikacijskih protokolov berljivih programerju, saj imajo strukturo protokolarnih sporočil običajno definirano s črkami abecede in ne več z zaporedjem posameznih bitov.

Druga skupina protokolov obsega lastniške protokole, katerih intelektualne pravice ohranljajo izključno njihovi razvijalci. Takšni protokoli imajo tudi lahko javne specifikacije (npr. Microsoft Exchange) ali pa ne (npr. protokol Skype). V primeru pomanjkanja javnih specifikacij poskušajo razvijalci omrežnih aplikacij ugotoviti delovanje teh protokolov z opazovanjem omrežnega prometa in postopkom *obratnega inženiringa* (angl. *reverse engineering*).

Pri razvoju omrežnih aplikacij je potrebno premisliti, kateri transportni protokol je najbolj primeren zanje. Spoznali smo namreč, da imata protokola TCP in UDP različne lastnosti. V kolikor aplikacija potrebuje zanesljivo dostavo, kontrolo pretoka in kontolo zasičenja, lahko sklepamo, da je zanjo bolj primerna raba povezavnega protokola TCP. Po drugi strani pa, če so pri komunikaciji pomembni čas prenosa, zakasnitev, potrebna minimalna pasovna širina, obenem pa lahko toleriramo izgube podatkov in zamenjan vrstni red datagramov, je lahko bolj primerna raba transportnega protokola UDP. Za opisane potrebe je protokol UDP primeren ravno zaradi njegove minimalističnosti, ki od udeležencev v komunikaciji zahteva zelo malo režije. Primerjavo lastnosti različnih aplikacij in izbire primernega transportnega protokola na podlagi teh lastnosti prikazuje tabela 6.1.

Opazimo, da med zahtevami aplikacij nikjer nismo omenili potreb po varnosti,

<sup>1</sup>Primer dokumenta RFC je RFC 2616, ki definira protokol HTTP/1.1: <https://tools.ietf.org/html/rfc2616>.

| aplikacija               | primeri aplikacijskih protokolov | toleranca izgube | potrebna min. pasovna širina | časovna občutljivost | običajni transportni protokol |
|--------------------------|----------------------------------|------------------|------------------------------|----------------------|-------------------------------|
| prenos datotek           | FTP                              | NE               | NE                           | NE                   | TCP                           |
| e-pošta                  | SMTP, POP3, IMAP                 | NE               | NE                           | NE                   | TCP                           |
| splet                    | HTTP                             | NE               | NE                           | NE                   | TCP                           |
| oddaljen dostop          | Telnet                           | DA               | 1-10 kb/s                    | DA                   | TCP                           |
| multimedija (realni čas) | RTP, HTTP                        | DA               | 10 kb/s - 5 Mb/s             | DA                   | TCP/UDP                       |
| multimedija (shranjena)  | RTP, HTTP                        | DA               | 10 kb/s - 5 Mb/s             | DA                   | TCP/UDP                       |
| igre                     | HTTP, lastniški                  | DA/NE?           | 1-10 kb/s                    | DA                   | TCP/UDP                       |
| IP telefonija            | SIP, RTP, lastniški              | DA               | 100 kb/s - 3 Mb/s            | DA                   | UDP                           |

**Tabela 6.1** Lastnosti različnih porazdeljenih aplikacij, na podlagi katerih temelji izbor ustreznega transportnega protokola

kot so avtentikacija, zakrivanje podatkov in zagotavljanje integritete aplikacijskih sporočil. Ker potreb po varnosti ne moremo zagotoviti s protokoloma TCP in UDP, lahko varno komunikacijo zagotovimo bodisi na omrežni plasti (protokoli IPSec) ali pa z uporabo protokolov na aplikacijski plasti (npr. SSL, TLS ali s protokoli, ki imajo že vgrajene naštete varnostne mehanizme, kot je npr. DNSSEC pri protokolu DNS).

## 6.1 Protokol HTTP

HTTP (angl. *HyperText Transfer Protocol*) je temeljni protokol, ki se uporablja v spletu (angl. *world wide web*). Pojavil se je v 1990. letih in omogočil delovanje spletu kot nove revolucionarne aplikacije, ki zagotavlja dostop do vsebin na zahtevo, iskanja, vključevanje grafičnih in multimedijskih gradiv ter ponuja objavljanje vsebin vsakomur.

Ker je HTTP javen in odprt protokol, je njegovo delovanje definirano v dokumentih RFC 1945 (HTTP 1.0) in RFC 2616 (novejša različica HTTP 1.1). V spletu je namenjen delovanju v arhitekturi odjemalec-strežnik, kjer strežnik običajno prejema zahtevke na vratih 80, odjemalec pa nanj naslavlja zahtevke (HTTP request) in prejema odgovore (HTTP response). Za zanesljivo dostavo, vrstni red, potrjevanje in ponovitve skrbi transportni protokol TCP.

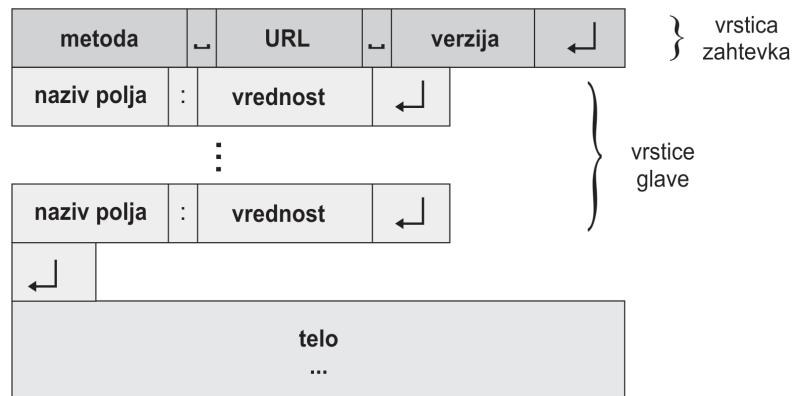
Za HTTP pravimo, da je protokol, ki *ne hrani stanja* (angl. *stateless*). To pomeni, da strežnik ne hrani evidence o preteklih zahtevkih s strani odjemalcev in njihovem številu. To lahko predstavlja težavo v primerih, kadar mora strežnik sprempljati zaporedje uporabnikovih dejanj (npr. posamezni koraki pri nakupu v

spletni trgovini), za kar pa tudi obstaja rešitev z uporabo piškotkov (angl. *cookies*), ki jo bomo spoznali ob koncu razdelka.

### 6.1.1 Oblike sporočil

Sporočilo **zahtevka HTTP** je prikazano na sliki 6.1. Vidimo, da ga tvori začetna vrstica zahtevka (angl. *request line*), sledijo pa ji več vrstic glave (angl. *header lines*), prazna vrstica in opcijsko telo sporočila (v primeru, če pošiljamo podatke na strežnik).

Vrstica zahtevka se začne s poljem *metoda*, ki definira dejanje, ki ga odjemalec zahteva od strežnika. Ima lahko vrednosti, prikazane v tabeli 6.2, od katerih pri uporabi močno prevladujeta metodi GET in POST.



Slika 6.1 Aplikacijsko sporočilo za **zahtevek HTTP**

Nadalje, vrstica zahtevka vsebuje še dve polji: polje *URL*, ki opredeljuje zahtevan objekt s strani strežnika in ga opredelimo z njegovim naslovom ter polje *verzija*, ki opredeljuje verzijo uporabljanega protokola in ima običajno vrednost HTTP/1.1. Vsa polja vrstice zahtevka so ločena s presledkom, vse vrstice zahtevka pa zaključuje kombinacija znakov za prehod v novo vrstico CR-LF (angl. carriage return-line feed).

Vrstici zahtevka lahko sledi več vrstic glave, ki podajajo množico parov v obliki ključ–vrednost. Imenu ključa, s katerim se vsaka vrstica začne, po dvopičju sledi njegova vrednost, vsaka vrstica pa je zaključena s kombinacijo znakov za prehod v novo vrstico. Poleg približno 40 različnih standardiziranih ključev, ki jih lahko opredelimo v vrsticah glave<sup>2</sup>, obstaja tudi veliko število nestandardiziranih, ki jih lahko uporabljajo izbrani spletni strežniki ali medstrežniki, ki izvajajo spletno predpomnenje. Od standardnih bomo v nadaljevanju omenili tri pogosto uporabljane, ki so *Host*, *Connection* in *If-Modified-Since*. Primer uporabe ključev *Host*

<sup>2</sup>Definirani v RFC 7230, 7231, 7232, 7233, 7234 in 7235.

| metoda  | opis                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                              |
|---------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| GET     | zahtevek po objektu (npr. besedilu spletnne strani, sliki ipd.)                                                                                                                                                                                                                                                                                                                                                                                                                                                   |
| POST    | zahtevek po objektu, ki istočasno strežniku pošilja podatke (pogosto uporabno v primeru pošiljanja vsebin polj v spletnem obrazcu, ki jih je vnesel uporabnik). Omenimo, da lahko podatke strežniku posredujemo sicer tudi z uporabo metode GET, pri čemer jih ne zapišemo v telo zahtevka, temveč kot nadaljevanje zahtevanega naslova URL za znakom "?" (npr. v naslovu <code>http://.../stran.html?lang=slv&amp;id=31318</code> sta <i>lang</i> in <i>id</i> parametra, katerih vrednosti pošiljamo strežniku) |
| HEAD    | zahteva po samo glavi odgovora brez vsebine zahtevanega objekta (uporabno za razhroščevanje)                                                                                                                                                                                                                                                                                                                                                                                                                      |
| PUT     | nalaganje vsebine na strežnik                                                                                                                                                                                                                                                                                                                                                                                                                                                                                     |
| DELETE  | brisanje vsebine s strežnika                                                                                                                                                                                                                                                                                                                                                                                                                                                                                      |
| TRACE   | zahtevek za razhroščevanje, ki vrne odmev (angl. <i>echo</i> ) poslanega zahtevka                                                                                                                                                                                                                                                                                                                                                                                                                                 |
| CONNECT | zahtevek za povezavo preko medstrežnika                                                                                                                                                                                                                                                                                                                                                                                                                                                                           |
| OPTIONS | povpraševanje o možnih opcijah pri zahtevku.                                                                                                                                                                                                                                                                                                                                                                                                                                                                      |

**Tabela 6.2** Seznam izbranih metod, ki jih lahko uporabimo v vrstici zahtevka

★ **Primer 6.1:** Primer zahtevka HTTP:

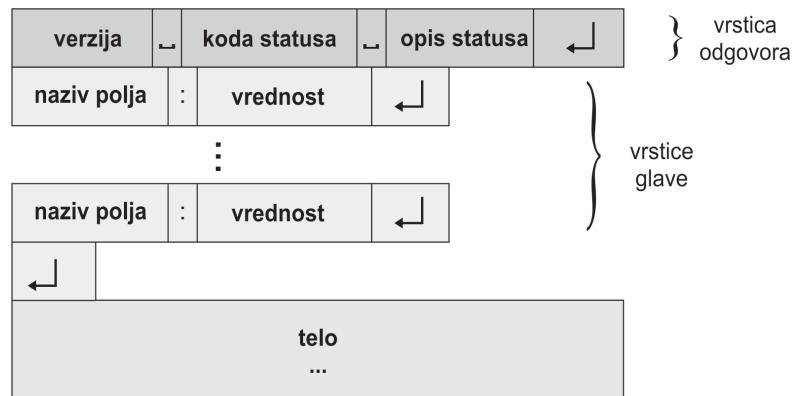
```
GET /online HTTP/1.1
Host: ldhbncxkztwsrfmv.neverssl.com
Connection: close
```

in *Connection* prikazuje tudi primer 6.1 zahtevka HTTP, ki dodatno opredeljuje naslov domene strežnika in zahtevo po rušenju povezave po prejetju odgovora.

Druga možna protokolarna oblika sporočila je **odgovor HTTP** (angl. *HTTP response*), ki ga strežnik vrne odjemalcu kot odgovor na prejeti zahtevek. S slike 6.2 lahko vidimo, da je struktura odgovora zelo podobna strukturi zahtevka, z nekaj razlikami v pomenu in vsebini posameznih polj. Statusno vrstico odgovora tvorijo polja *verzija*, ki podaja verzijo uporabljenega protokola HTTP, in polji *koda statusa* ter *opis statusa*, ki podajata numerično in opisno kodo statusa strežnikovega odgovora. Kode in opise statusov<sup>3</sup>, ki so prikazani tudi v tabeli 6.3, delimo v pet skupin glede na njihovo resnost. Skupine različnih statusov so grupirane glede na prvo števko v trimestri kodi odgovora.

Poleg statusne vrstice odgovor HTTP vsebuje tudi vrstice glave, ki so v enaki zapisni obliki kot vrstice glave zahtevka, in opcijsko telo, ki vsebuje podatke, s katerimi odgovori strežnik. Tudi vrstice glave odgovora HTTP delimo v standardizirane in nestandardizirane, opredeljene pa so v istih virih kot zahtevki HTTP. Od

<sup>3</sup>Opredeljene so v RFC 7231



Slika 6.2 Aplikacijsko sporočilo za odgovor HTTP

★ **Primer 6.2:** Primer glave odgovora HTTP:

```
HTTP/1.1 200 OK
Content-Type: text/html
Content-Length: 2778
Connection: close
Date: Mon, 05 Nov 2007 12:18:23 GMT
Server: Apache/1.3.0 (Unix)
Last-Modified: Fri, 16 Feb 2018 21:32:40 GMT
```

njih omenimo morda samo najbolj pogoste, ki so: *Connection* (status povezave), *Date* (datum in čas pošiljanja odgovora), *Server* (ime strežnika), *Last-Modified* (čas zadnje spremembe prejetega vira), *Content-Length* (dolžina vira v bajtih) in *Content-Type* (tip MIME vsebine, npr. *text/html*, *image/jpeg*, *audio/mpeg* ipd.). Vzorec odgovora HTTP je prikazan v primeru 6.2.

### 6.1.2 Vrste povezav

Iz vrstic glave v zahtevku HTTP smo videli, da lahko odjemalec od strežnika zahteva prekinitev povezave s ključem *Connection: Close*, čemur pravimo, da odjemalec izvede **neveztrajno** (angl. *non-persistent*) povezavo. Ker vemo, da se za izvedbo zahtevka HTTP vzpostavi povezava TCP, postopek trojnega rokovanja pri njeni vzpostavitvi tudi zahteva svoj čas. Iz tega sledi, da se za vsak prenašani vir s strežnika (stran z besedilom, sliko ipd.) porabi 2 RTT časa (1 RTT za rokovanje + 1 RTT za prenos; predpostavimo, da lahko odjemalec v tretji fazi rokovanja že pošlje podatke po principu *na štuporamo*).

Alternativa neveztrajni povezavi bi lahko bila, da odjemalec ob pošiljanju zahtevka ne zahteva rušenja povezave (izpusti ključ *Connection: Close*) in s tem

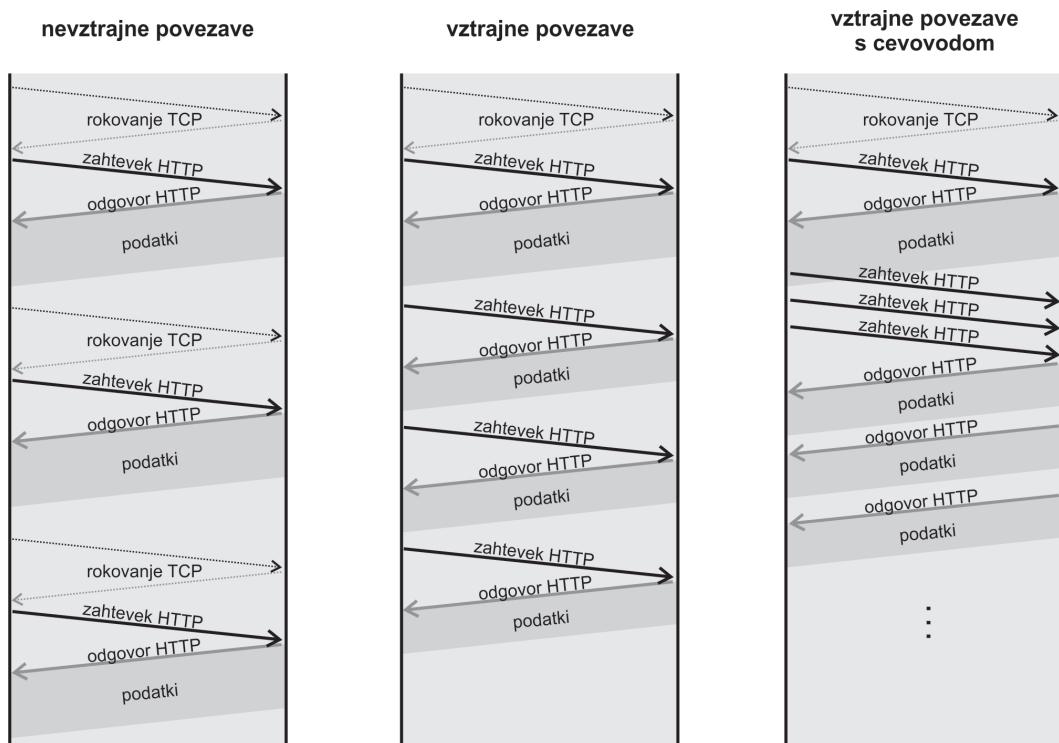
| KODA       | OPIS STATUSA                      |                                                                                                                 |
|------------|-----------------------------------|-----------------------------------------------------------------------------------------------------------------|
| <b>1xx</b> | <b>INFORMATIVNO</b>               |                                                                                                                 |
| 100        | Continue                          | strežnik je prejel glavo in sporoča, da pričakuje še telo zahtevka                                              |
| 102        | Processing                        | strežnik sporoča, da še vedno obdeluje zahtevek (zaradi preobremenjenosti)                                      |
| <b>2xx</b> | <b>USPEH</b>                      |                                                                                                                 |
| 200        | OK                                | standarden odgovor za uspešen zahtevek                                                                          |
| <b>3xx</b> | <b>PREUSMERITEV</b>               |                                                                                                                 |
| 301        | Moved Permanently                 | sporočilo, da mora biti zahteva naslovljena na drugi podani naslov                                              |
| 304        | Not Modified                      | sporoča, da zahtevan vir ni bil spremenjen od podanega datuma (uporabno pri medstrežnikih)                      |
| 305        | Use Proxy                         | zahtevan vir je na razpolago samo pri uporabi medstrežnika                                                      |
| <b>4xx</b> | <b>NAPAKA NA STRANI KLIENTA</b>   |                                                                                                                 |
| 400        | Bad Request                       | strežnik ne razume zahtevka, verjetno zaradi napačne sintakse                                                   |
| 403        | Forbidden                         | dostop do vira je prepovedan, verjetno zaradi pomanjkanja pravic za dostop do strežnikovega datotečnega sistema |
| 404        | Not Found                         | vira ni mogoče najti                                                                                            |
| 408        | Request Timeout                   | potek časovne kontrole za to, da klient v celoti dokonča zahtevek                                               |
| 429        | Too Many Requests                 | uporabnik je v časovni enoti poslal preveliko število zahtevkov                                                 |
| <b>5xx</b> | <b>NAPAKA NA STRANI STREŽNIKA</b> |                                                                                                                 |
| 500        | Internal Server Error             | generično sporočilo za napako na strežniku, ko je strežnik ne more podrobneje opredeliti                        |
| 501        | Not Implemented                   | strežnik nima implementirane metode, opredeljene v zahtevku                                                     |
| 502        | Bad Gateway                       | strežnik, ki opravlja funkcijo medstrežnika, je od tretjega strežnika prejel neveljaven odgovor                 |
| 505        | HTTP Version Not Supported        | verzija protokola HTTP, uporabljena v zahtevku, ni podprta na strežniku                                         |

**Tabela 6.3** Najbolj pogoste kode in opisi statusov v odgovorih HTTP

zahteva **vztrajno povezavo** (angl. *persistent*), oziroma da povezava ostane po prejemu odgovora še vedno odprta. V nadaljevanju lahko odjemalec in strežnik uporabita isto povezavo za pošiljanje morebitnih nadaljnjih virov, ki so lahko del iste spletne strani (npr. slike s spletne strani), s čimer se izognemo ponovni potrebi po rokovovanju pri protokolu TCP. To pomeni, da prenos vseh odgovorov razen prvega lahko izvedemo že v času 1 RTT, kar predstavlja bistveno pohitritev kot pri nevztrajnih povezavah. Iz poznавanja nadzora zasičenja protokola TCP obenem vemo, da protokol povečuje hitrost pošiljanja, kar vodi v dodatno prednost dlje časa odprte povezave, ki lahko čas prenosa še dodatno pohitri, kar vidimo na sliki 6.3.

Poleg navadnih vztrajnih povezav poznamo tudi njihovo posebno obliko: **vztrajne povezave s cevovodom** (angl. *pipelined persistent*). Te omogočajo

pošiljanje več vzporednih zahtevkov s strani odjemalca strežniku naenkrat in kasnejše prejemanje ustreznih odgovorov. Takšen način pošiljanja smo spoznali že kot *tekoče pošiljanje* na transportni plasti. Enako kot tam, morata tukaj odjemalec in strežnik na aplikacijski plasti voditi seznam poslanih in že potrjenih aplikacijskih zahtev, ki so neodvisne od potrjevanja na transportni plasti.



Slika 6.3 Vrste povezav pri protokolu HTTP

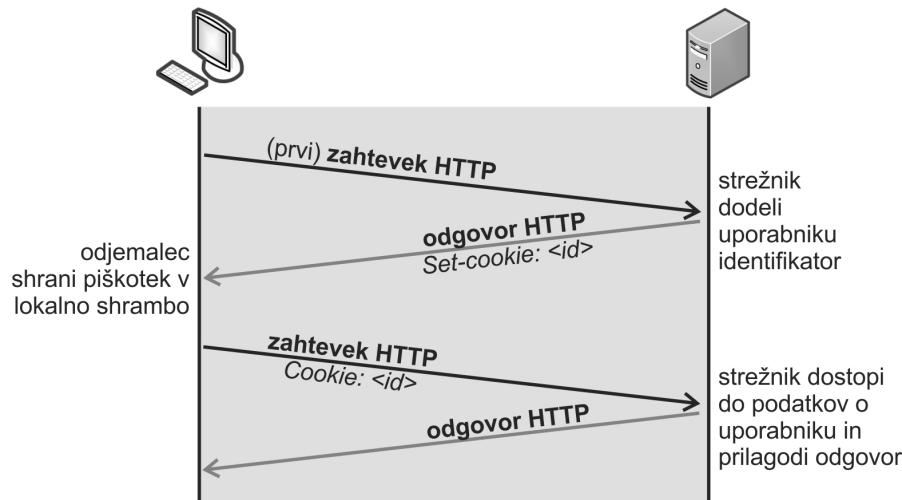
### 6.1.3 Uporaba piškotkov

V uvodnem opisu protokola HTTP smo omenili, da ta ne hrani stanja povezav (angl. *stateless*), kar pomeni, da strežnik ne vodi evidence o vsebini in številu preteklih uporabnikovih zahtev. Delovanje spletnih strežnikov je na ta način preprosto, vendar pa v taki obliki ne podpira sodobnih spletnih aplikacij, ki za doseganje boljše uporabniške izkušnje potrebujejo tudi informacijo o stanju. Potrebe po ukrepanju glede na predhodne akcije uporabnika lahko namreč vidimo v naslednjih primerih spletnih aplikacij:

- **spletne trgovine:** uporabniku je potrebno slediti skozi nakupovalne korake in npr. v nakupovalni košarici prikazati artikle, ki si jih je izbral v prejšnjih korakih,

- **personalizacija:** če si uporabnik nastavi posebne nastavitev prikaza spletnega mesta (npr. barva ozadja, oblika spletnega vmesnika), je potrebno, da ga strežnik identificira in mu prikaže njegovo personalizirano vsebino,
- **priporočila:** da spletna stran prikaže uporabniku priporočila o sorodnih artiklih ali zanimivih filmih, mora spremljati zgodovino uporabnika skozi katero analizira, kaj uporabnika zanima,
- **avtorizacija:** da uporabniku omogočimo dostop do spletnih virov, ga je predhodno potrebno avtenticirati in na podlagi njegove identitete sproti preverjati pravice dostopa do zahtevanih spletnih virov.

Problem pomanjkanja podatkov o stanju pri protokolu HTTP lahko rešimo z uporabo **piškotkov** (angl. *cookies*). Piškotki so kratki nabori podatkov, ki lahko hranijo podatke v strukturi ključ-vrednost, poleg njih pa tudi identifikator, ki ga strežnik dodeli odjemalcu. Strežnik dodeli odjemalcu piškotek z uporabo polja *Set-Cookie* v glavi strežnikovega odgovora HTTP. Po prejetju piškotka ga odjemalec običajno shrani v svojem datotečnem sistemu, do katerega dostopa tudi brskalnik. Odjemalec nadalje pri vsaki komunikaciji s strežnikom v glavo zahtevka pripne piškotek v polje *Cookie*, s čimer ga strežnik torej lahko ponovno prepozna. Ker strežnik ve, da gre za znanega uporabnika, lahko svoje odgovore prilagodi zgodovini preteklih zahtevkov. Pri tem strežnik uporablja tudi podatkovno bazo, v kateri vzdržuje potrebne podatke o uporabniku za izvedbo aplikacije, ki potrebuje sledenje stanju uporabnika. Primer delovanja piškotkov prikazuje slika 6.4.



**Slika 6.4** Uporaba piškotkov pri protokolu HTTP

~~~~~ **Varnostni pomislek:** Ker piškotke lahko uporabljamo za hranjenje podatkov o uporabniku, se je veljavno vprašati, ali so vsi nameni uporabe piškotkov

dobronamerni. Žal temu ni tako, saj zbiranje podatkov o uporabnikih lahko odpira tudi etična vprašanja, kot je poseganje v uporabnikovo zasebnost. Eden bolj razvitetih primerov zlorabe piškotkov spada na področje spletnega oglaševanja, kjer se z vodenjem evidence o uporabnikovi zgodovini obiskanih strani in evidence o uporabljenih iskalnih nizih lahko uporabnike profilira z namenom ciljanega oglaševanja. Z namenom preprečevanja zlorab lahko v splettem brskalniku onemogočimo uporabo piškotkov, v zadnjih letih pa smo bili priča tudi uvedbi zakonodaje, ki zahteva informiranje uporabnika o namembnosti piškotov na spletni strani in zahteva njegovo soglasje.

6.1.4 Uporaba spletnih medstrežnikov

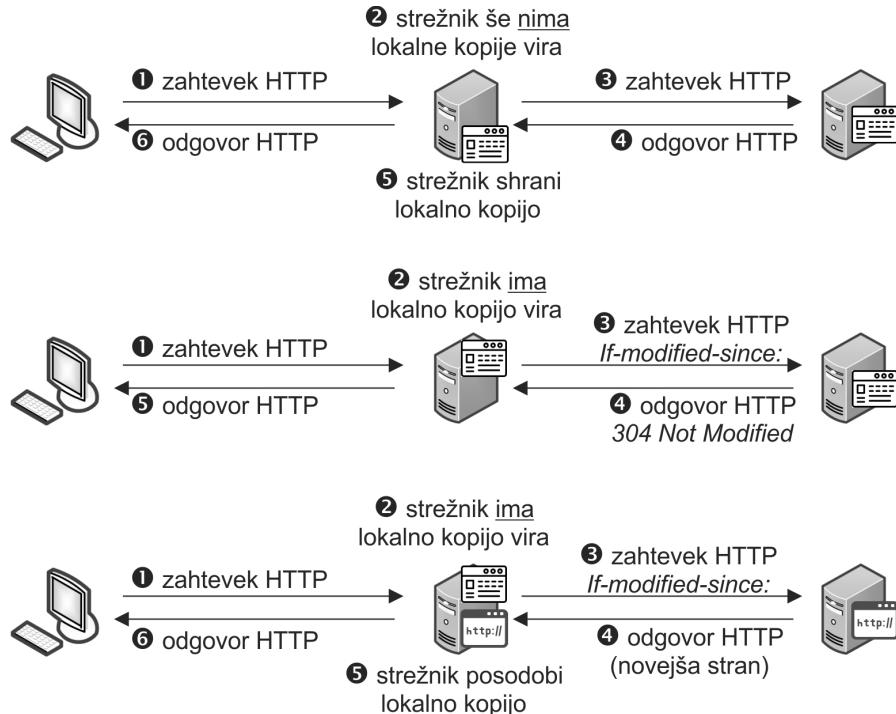
Spletni medstrežniki (angl. *proxy server*) so strežniki, s katerimi lahko zagotovimo hitrejši dostop do vsebin na spletu. Njihov namen je, da igrajo vlogo posrednika med odjemalcem in strežnikom in pri sebi hranijo kopije spletnih virov. Če medstrežnik pri sebi kopije zahtevanega vira še nima, jo pridobi z dostopom do pravega strežnika; v primeru, če pa kopijo že ima, jo odjemalcu takoj postreže brez nadaljnjega dostopa v svetovni splet. Na ta način medstrežnik nenehno igrat vlogo strežnika proti odjemalcu in vlogo odjemalca proti pravemu strežniku na spletu.

Iz opisanega izhaja, da morajo odjemalci biti ustrezno konfigurirani, saj morajo svoje zahtevke HTTP posredovati medstrežniku in ne več neposredno strežnikom na spletu. Uporabo medstrežnikov lahko bodisi ročno nastavimo v splettem brskalniku ali pa administrator omrežja poskrbi za samodejno nastavitev brskalnikov.

Medstrežnike lahko bodisi namestimo v lokalno omrežje (če je večje) ali pa nam jih v rabo običajno ponuja ponudnik internetnih storitev. Obe opisani postaviti medstrežnika zagotavlja, da je ta krajevno blizu, saj to omogoča hitrejši dostop (oz. manjšo zakasnitev) do spletnih vsebin kot dostop do poljubnega strežnika kjerkoli na svetu. Namestitev medstrežnika v lokalno omrežje obenem zagotavlja manj prometa na dostopni povezavi do javnega omrežja, kar lahko predstavlja tudi prihranke v primeru zakupa povezave glede na količino podatkov.

Medstrežniki izvajajo svoje poizvedbe tako kot prikazuje slika 6.5. V primeru, da medstrežnik zahtevanega spletnega vira še nima, naslovi na končni strežnik običajni zahtevek, s katerim pridobi lastno kopijo zahtevanega vira. V nasprotnem primeru pa, če medstrežnik že hrani lokalno kopijo oddaljenega vira, medstrežnik izvede *pogojni zahtevek GET* (angl. *conditional GET*). V tem zahteveku medstrežnik uporabi vrstico glave *If-modified-since* (primer: *If-modified-since: Wed, 12 Feb 2012 09:12:42*), ki ji sledi datum lokalne verzije vira, ki ga zahteva. S tem opredeli, da naj končni strežnik vir posreduje le v primeru, če je njegova različica novejša od lokalno shranjene različice. Končni strežnik lahko na pogojni zahtevek odgovori bodisi s pošiljanjem novejšega vira (status 200 OK) ali pa z

odgovorom s statusom 304 *Not Modified*, ki ima prazno telo in torej varčuje pri ponovnem pošiljanju nepotrebnih podatkov.



Slika 6.5 Vloga medstrežnika HTTP pri poizvedbah HTTP. Prikazani so trije scenariji: strežnik pridobi lokalno kopijo strani (zgoraj), strežnik ugotovi, da že ima aktualno kopijo strani (sredina), strežnik posodobi lokalno kopijo z novejšo (spodaj).

6.2 Protokol FTP

FTP (angl. *File Transfer Protocol*)⁴ je protokol, ki se uporablja za dvosmeren prenos datotek preko omrežja med oddaljenim in lokalnim datotečnim sistemom. Podobno kot HTTP predstavlja tudi FTP enega pionirskeih protokolov, katerega zametki so prisotni že od 70. let prejšnjega stoletja. Za razliko od protokola HTTP mora strežnik FTP hraniti stanje uporabnika, saj mora uporabnika spremljati skozi postopek avtentifikacije, nadzorovati avtorizacijske pravice in slediti gibanju uporabnika skozi mape v oddaljenem datotečnem sistemu.

Delovanje protokola FTP poteka ravno tako na osnovi zahtevkov in odgovorov. Aplikacijska sporočila odjemalca tipično vsebujejo zahteve po prenosu datotek, ukaze za posredovanje avtentikacijskih podatkov in ukaz za zaključek seje

⁴RFC959, RFC1639, RFC2228

(seznam najbolj pogostih ukazov je prikazan v tabeli 6.4). Na zahtevke strežnik odgovori s statusom odgovora (najbolj pogoste lahko vidimo v tabeli 6.5) in dodatnimi podatki, če so zahtevani.

| ukaz | opis |
|-------------------|--|
| USER ime | posredovanje uporabniškega imena |
| PASS geslo | posredovanje gesla |
| LIST | zahteva po seznamu datotek v mapi |
| CWD ime | zamenjava trenutne mape |
| RETR ime_datoteke | zahteva po prenosu datoteke s strežnika |
| STOR ime_datoteke | zahteva po shranjevanju datoteke na strežnik |
| DELE ime_datoteke | zahteva po brisanju oddaljene datoteke |
| QUIT | zaključek seje |

Tabela 6.4 Seznam izbranih ukazov (zahtevkov) protokola FTP

| koda statusa | opis |
|--------------|---|
| 125 | Data connection open, transfer starting |
| 200 | Command okay. |
| 226 | Closing data connection. |
| 227 | Entering Passive Mode <h1,h2,h3,h4,p1,p2> |
| 230 | User logged in, proceed. |
| 331 | Username OK, password required |
| 332 | Need account for login. |
| 425 | Can't open data connection. |
| 426 | Connection closed; transfer aborted. |
| 452 | Error writing file |
| 425 | Can't open data connection |
| 500 | Syntax error, command unrecognized. |
| 530 | Not logged in. |
| 552 | Requested file action aborted. |

Tabela 6.5 Seznam izbranih statusov v odgovoru protokola FTP

Posebnost protokola FTP je ta, da med odjemalcem in strežnikom tečeta dve vzporedni ločeni povezavi TCP: *kontrolna* in *podatkovna* povezava. Kontrolna povezava se vzpostavi na začetku na pobudo odjemalca, in sicer tako, da odjemalec vzpostavi povezavo TCP s strežnikom preko vrat 21, na katerih strežnik aktivno posluša. Ob vzpostavitvi kontrolne povezave odjemalec posreduje tudi (z ukazom PORT) svojo številko vrat, na katerih je pripravljen vzpostaviti podatkovno povezavo. Strežnik nato vzpostavi podatkovno povezavo s svojih izhodnih TCP vrat 20 na naslov, ki ga je opredelil odjemalec z ukazom PORT. V nadaljevanju se kontrolna povezava uporablja za prenos kontrolnih podatkov (ukazi za prenos datotek, posredovanje uporabniškega imena/gesla, menjavo map, strežnikov status odgovora itd.), podatkovna povezava pa za prenašano zahtevano vsebino (prenos podatkov, prenos seznama datotek v mapi itd.). Primer komunikacije z uporabo protokola FTP lahko vidimo v primeru 6.3.

Pri opisanem načinu vzpostavljanja podatkovne povezave lahko nastopi težava, če je odjemalec skrit v omrežju za usmerjevalnikom NAT. Če odjemalec

★ **Primer 6.3:** Primer komunikacije s strežnikom FTP, kjer smo pri uporabi klienta uporabili tudi stikalo -d (debug), kar nam omogoči vpogled v aplikacijske zahtevke, ki jih klient pošilja strežniku (njihove vrstice se začnejo z zaporedjem --->). Iz komunikacije je razvidna prijava z uporabniškim imenom anonymous, pregled vsebine mape, izvedba ukazov PORT, prenos datoteke na lokalni računalnik in zaključek seje.

```
> ftp -d speedtest.tele2.net
Connected to speedtest.tele2.net.
220 (vsFTPd 3.0.3)
---> OPTS UTF8 ON
200 Always in UTF8 mode.
User (speedtest.tele2.net:(none)): anonymous
---> USER anonymous
331 Please specify the password.
Password:
---> PASS
230 Login successful.
ftp> dir
---> PORT 192,168,1,132,228,38
200 PORT command successful. Consider using PASV.
---> LIST
150 Here comes the directory listing.
-rw-r--r-- 1 0 0 1024 Feb 19 2016 1KB.zip
-rw-r--r-- 1 0 0 1048576 Feb 19 2016 1MB.zip
-rw-r--r-- 1 0 0 209715200 Feb 19 2016 200MB.zip
-rw-r--r-- 1 0 0 20971520 Feb 19 2016 20MB.zip
-rw-r--r-- 1 0 0 2097152 Feb 19 2016 2MB.zip
-rw-r--r-- 1 0 0 3145728 Feb 19 2016 3MB.zip
-rw-r--r-- 1 0 0 524288000 Feb 19 2016 500MB.zip
226 Directory send OK.
ftp: 1208 bytes received in 0.19Seconds 6.26Kbytes/sec.
ftp> get 1KB.zip
---> PORT 192,168,1,132,228,50
200 PORT command successful. Consider using PASV.
---> RETR 1KB.zip
150 Opening BINARY mode data connection for 1KB.zip (1024 bytes).
226 Transfer complete.
ftp: 1024 bytes received in 0.20Seconds 5.04Kbytes/sec.
ftp> quit
---> QUIT
221 Goodbye.
```

prvi ne vzpostavi podatkovne povezave, je strežnik z vrat 20 do njega ne bo mogel vzpostaviti, saj usmerjevalnik še nima zapisa za dostop do odjemalčevih vrat, ki so namenjena podatkovni povezavi. Da bi protokol FTP lahko deloval tudi v takšnih okoliščinah, obstaja tudi njegov spremenjen način delovanja, ki ga imenujemo **pasivni način**. V pasivnem načinu klient še vedno prvi vzpostavi kontrolno povezavo s strežnikovimi vrati 21. V nadaljevanju pa, namesto da pošlje strežniku ukaz PORT, mu pošlje ukaz PASV. Ob prejemu tega ukaza strežnik odpre naključna izhodna vrata (> 1024) in njihovo številko posreduje odjemalcu po kontrolni povezavi kot odgovor na ukaz PASV. Odjemalec je sedaj lahko tisti, ki se poveže z navedenimi vrati strežnika in prvi poskrbi tudi za vzpostavitev podatkovne povezave. Ob njeni vzpostavitvi se v usmerjevalnikovo tabelo NAT zapišejo ustrezni vnos, tako da lahko strežnik odjemalcu pošilja tudi povratni promet.

6.3 Protokoli za elektronsko pošto

Poleg spletja je elektronska pošta danes najbolj razširjena in uporabljana porazdeljena storitev, saj omogoča izredno hiter, poceni in postopkovno enostaven prenos sporočil. Poleg prenosa golih besedilnih sporočil omogoča tudi prenos oblikovanih sporočil (HTML), spletnih povezav in datotečnih pripomBK.

Protokoli za elektronsko pošto ravno tako delujejo na principu odjemalec-strežnik. Arhitektura porazdeljene aplikacije vsebuje:

- poštne strežnike, ki bodisi skrbijo za pošiljanje elektronskih poročil ali shranjujejo prejeto pošto v poštnih predalih uporabnikov,
- odjemalske programe, ki imajo lahko bodisi besedilni (npr. Elm, Pine) ali grafični uporabniški vmesnik (npr. Microsoft Outlook, Mozilla Thunderbird, Apple Mail, Eudora, Elm),
- aplikacijske protokole za pošiljanje novih (SMTP) in prebiranje dostavljenih elektronskih sporočil (POP3, IMAP4).

Tipičen življenjski cikel elektronskega sporočila je tak, da ga uporabnik sestavi z uporabo aplikacije (e-mail odjemalca) in ga nato pošlje lokalnemu poštnemu strežniku z uporabo protokola SMTP. Strežnik uvrsti sporočilo v izhodno čakalno vrsto za pošiljanje, in ko pride na vrsto, vzpostavi povezavo (neposredno ali preko verige drugih strežnikov) s ciljnim poštnim strežnikom, na katerem se nahaja elektronski predal ciljnega prejemnika elektronskega sporočila. Dokler se sporočilo hrani na strežniku uporabnika, lahko prejemnik uporabi svoj odjemalski program za prenos sporočila lokalno k sebi (npr. z uporabo protokolov POP3 in IMAP4) in njegov prikaz.

6.3.1 Protokol SMTP

Protokol SMTP (angl. *Simple Mail Transfer Protocol*) je tudi eden od začetnih aplikacijskih protokolov Interneta, saj je bil razvit že leta 1982⁵. Njegova specifikacija je bila leta 2008 dopolnjena⁶ z dodatki, ki vključujejo tudi varnostne mehanizme, celoten protokol pa danes imenujemo ESMTP (angl. *Extended SMTP*).

Tudi protokol SMTP je namenjen delovanju v načinu odjemalec-strežnik. Strežnik običajno posluša na TCP vratih 25, na katerem sprejema ukaze in podatke (elektronska sporočila), ki pa morajo biti zakodirano v kodno tabelo znakov 7-bitno ASCII⁷. Ker so dokumenti in slike, ki jih običajno pošiljamo kot priponke, kodirane z binarnim kodiranjem (različno dolga zaporedja bitov predstavlja kodirane objekte in ne znake), to pomeni, da moramo mora pošiljatelj binarne priponke pred pošiljanjem prekodirati v 7-bitni ASCII, prejemnik pa to storiti v obratno smer. Ker kodiramo iz bolj obsežnega predstavitevnega nabora znakov v 7-bitni nabor (128 znakov) to pomeni, da se pri tem dolžina priponk običajno poveča. Pogosti ukazi, ki jih naslovimo strežniku, so prikazani v tabeli 6.6, statusi strežnikovih odgovorov pa v tabeli 6.7.

| ukaz | opis |
|------------|---|
| HELO | predstavimo se strežniku z opredelitvijo imena lastne domene |
| MAIL FROM: | podamo naslov pošiljatelja |
| RCPT TO: | podamo naslov prejemnika; ukaz lahko uporabimo večkrat, če imamo več prejemnikov |
| DATA | pričetek pošiljanja podatkov (vsebine sporočila); pošiljanje zaključimo z zaporedjem <prehod v novo vrstico>. (pika)<prehod v novo vrstico> |
| QUIT | zaključek seje s strežnikom |

Tabela 6.6 Seznam izbranih ukazov (zahetvkov) protokola SMTP

Elektronsko sporočilo, ki ga odjemalec vnese po ukazu DATA, je sestavljeno iz vrstic glave, prazne vrstice in telesa (vsebine) sporočila. Podobno kot pri protokolu HTTP, glava hrani množico parov ključ-vrednost, ki opredeljujejo dodatne podatke aplikacijskemu odjemalcu za prikaz sporočila končnemu uporabniku. Glava običajno vsebuje vsaj polje From: in To:, ki opredeljujeta tudi polni osebni imeni pošiljatelja in prejemnika, in polje Subject:, ki podaja zadevo sporočila. Primer komunikacije s strežnikom in zapisa glave prikazuje primer 6.4.

Za zaključek lahko naredimo še kratko kvalitativno primerjavo protokolov protokola SMTP z že znanim protokolom HTTP, s katero lahko poudarimo bistvene razlike v principih delovanja:

⁵RFC 821

⁶RFC 5321

⁷Kodna tabela 7-bitni ASCII vsebuje predstavitev 128 znakov, od katerih je 33 kontrolnih znakov in 95 preostalih znakov in ločil, ki vsebujejo: !"#\$%&'()*+,-./0123456789:;=>?@ABCDEFGHIJKLMNPQRSTUVWXYZ[\]^_`abcdefghijklmnopqrstuvwxyz|~

| koda statusa | opis |
|--------------|--|
| 220 | Server is ready. |
| 221 | Closing connection. |
| 250 | Requested mail action okay, completed. |
| 354 | This is a reply to the DATA command. After getting this, start sending the body of the mail message, ending with "\r\n.\r\n" |
| 500 | The last command contained a syntax error or the command line was too long. |
| 503 | The last command was sent out of sequence. For example, you might have sent DATA before sending RECV. |
| 552 | The recipient mailbox is full. Try again later. |
| 553 | The mail address that you specified was not syntactically correct. |

Tabela 6.7 Seznam izbranih statusov v odgovoru protokola SMTP

★ **Primer 6.4:** Primer komunikacije s strežnikom SMTP. Iz sporočila lahko razberemo ukaze odjemalca (HELO, MAIL FROM, RCPT TO, DATA, vsebino sporocila, QUIT) in strežnikove odgovore (vrstice statusov, ki se pričnejo z numerično kodo). Primer prikazuje izmišljena imena strežnikov, domen, pošiljatelja in prejemnika.)

```

220 postni.streznik.si ESMTP
HELO domena.com
250 postni.streznik.si
MAIL FROM:moje.ime@domena.com
250 2.1.0 Ok
RCPT TO:ime.prejemnika@druga.domena.com
250 2.1.5 Ok
DATA
354 End data with <CR><LF>.<CR><LF>
From: Ime Priimek <moje.ime@domena.com>
To: Prejemnik <ime.prejemnika@druga.domena.com>
Subject: Kratko sporocilo

To je e-mail sporocilo brez priponke.

.
250 2.0.0 Ok: queued as C29CE4010A
QUIT
221 2.0.0 Bye

```

- medtem, ko protokol HTTP deluje s principom *pull* (prenos podatkov – spletnih virov s strežnika k odjemalcu), deluje protokol SMTP s principom *push* (prenos podatkov – elektronskega sporočila od odjemalca k strežniku),
- za razliko od protokola SMTP, ki zahteva kodiranje podatkov v naboru 7-bitni ASCII, lahko protokol HTTP uporablja tudi binarno kodiranje,
- medtem, ko se pri protokolu HTTP vsak objekt pošilja ločeno v svojem aplikacijskem sporočilu, se pri protokolu SMTP lahko več objektov (sporočila, priponke) prenese v istem sporočilu,

- protokol HTTP lahko uporablja bodisi vztrajne ali nevztrajne povezave, medtem ko so povezave pri protokolu SMTP vedno vztrajne.

6.3.2 Protokol POP

Za razliko od protokola SMTP, ki se uporablja za pošiljanje elektronske pošte, obstajajo tudi protokoli, ki omogočajo uporabniku dostop do svojega elektronskega predala na strežniku in prenos tamkajšnjih elektronskih sporočil v lokalne kopije. Primeri najpogostejših protokolov za takšen dostop do elektronske pošte sta POP (angl. *Post Office Protocol*)⁸ in IMAP (angl. *Internet Message Access Protocol*)⁹. Protokol POP je preprosteeji in omogoča osnovne funkcije, kot so avtentikacija uporabnika pri dostopu do elektronskega predala, pregledovanje in prenos pošte s strežnika. Protokol IMAP je mlajši in nekoliko naprednejši, poleg funkcionalnosti protokola POP pa omogoča tudi samo prenos elektronskih sporočil, organizacijo sporočil po mapah in bolj zanesljivo vzporedno delo s strežnikom s strani več hkratnih odjemalcev (slednje sicer zvišuje obremenjenost strežnika). Omenimo še, da v sodobnem času do elektronske pošte dostopamo pogosto tudi kar preko spletnih odjemalcev, ki jih uporabljamo v spletnem brskalniku (npr. Gmail, Yahoo!, Outlook, Mail.com, AOL ipd.). Tak dostop do odjemalčevega aplikacijskega vmesnika sicer deluje na podlagi protokola HTTP, a v ozadju lahko te spletne aplikacije uporabljajo tradicionalne protokole za dostop do spletnih strežnikov podjetja, kot je tudi protokol POP.

Strežnik protokola POP deluje običajno na TCP vratih 110. Dialog med odjemalcem in strežnikom tipično poteka v treh fazah: (1) avtentikacija, (2) prenos sporočil in (3) zaključek dialoga in posodabljanje elektronskega predala. Tako kot protokola HTTP in SMTP tudi protokol POP ne hrani informacije o stanju, deluje pa s principom *pull* (prenos podatkov s strežnika k odjemalcu). Pogosti aplikacijski zahtevki odjemalca so prikazani v tabeli 6.8, odgovori strežnika nanje pa so izredno preprosti, saj strežnik večinoma odgovori z "+OK" ali napako "-ERR pojasnilo", ki podaja razlog za neizvedbo operacije (npr. neznano uporabniško ime, napačno geslo, neobstoječe elektronsko sporočilo).

Tipičen potek dialoga med odjemalcem in strežnikom je prikazan na primeru 6.5. Iz vsebine sporočila v tem primeru lahko vidimo, kako aplikacijski odjemalci in strežniki, preko katerih je sporočilo potovalo, dodajo številne dodatne vrstice glave. Med njimi najbolj prevladujejo vrstice `Received:`, ki beležijo, kdaj je kateri strežnik na poti sporočilo prevzel.

⁸RFC 1939

⁹RFC 1739

★ **Primer 6.5:** Primer dialoga me odjemalcem in strežnik POP. Iz sporočila lahko razberemo ukaze odjemalca (USER, PASS, LIST, RETR, DELE in QUIT) in strežnikove odgovore (statuse in vsebino elektronskega sporočila). Za potrebe objave v knjigi so bila dejanska imena strežnikov, domen, pošiljatelja in prejemnika zamenjena z izmišljenimi.)

```
+OK Server ready.
USER gandalf
+OK
PASS ushal1n0tpass
+OK Logged in.
LIST
+OK 5 messages:
1 2279
2 2256
3 2357
4 12582
5 12375
.
RETR 5
+OK 3356 octets
Return-Path: <bilbo@lotr.com>
Delivered-To: <gandalf@lotr.com>
Received: from localhost (localhost [127.0.0.1]) by mail.fri.uni-lj.si (Postfix)
          with ESMTP id 2150540120 for <gandalf@lotr.com>; Wed, 8 Apr 2020
          13:14:28 +0200 (CEST)
Received: from mail.fri.uni-lj.si ([127.0.0.1]) by localhost (mail.fri.uni-lj.si
          [127.0.0.1]) (amavisd-new, port 10024) with ESMTP id 73JgSsF87D09 for
          <gandalf@lotr.com>; Wed, 8 Apr 2020 13:14:26 +0200 (CEST)
Received: from mail-FRI.fri1.uni-lj.si (unknown [212.235.188.21]) by
          mail.fri.uni-lj.si (Postfix) with ESMTPS id CD24B4010C for
          <gandalf@lotr.com>; Wed, 8 Apr 2020 13:14:26 +0200 (CEST)
<... skrajšan izpis glave elektronskega sporočila ... >
To: "gandalf@lotr.com" <gandalf@lotr.com>
Subject: Lep pozdrav
Thread-Topic: Lep pozdrav
Thread-Index: AdYNltz9bY69R1JLQLWw01hHP9Y4cg==
Date: Wed, 8 Apr 2020 11:14:26 +0000
Message-ID: <bdef88babdfc4a9faeaf3ef0a2a90ef9@fri.uni-lj.si>

Moj prvi e-mail!
.
DELE 5
+OK Marked to be deleted.
QUIT
+OK Logging out, messages deleted.
```

| ukaz | opis |
|-----------------|---|
| USER <i>u</i> | podajanje uporabniškega imena <i>u</i> za dostop do strežnika |
| PASS <i>p</i> | podajanje pripadajočega gesla <i>p</i> za uporabniško ime |
| LIST | izpis seznama sporočil v obliki (# sporočila, velikost) |
| RETR # <i>n</i> | prenos sporočila s številko # <i>n</i> |
| DELE # <i>n</i> | brisanje sporočila s številko # <i>n</i> |
| STAT | statistika o številu in velikosti vseh sporočil |
| RSET | ponastavitev elektronskega predala na stanje ob začetku seje (preklic brisanja pobrisanih sporočil) |
| NOOP | operacija brez učinka, ohranja vzpostavljeno povezavo |
| QUIT | zaključek seje s strežnikom |

Tabela 6.8 Seznam izbranih ukazov (zahtevkov) protokola POP

★ **Primer 6.6:** Kot analogijo iz realnega sveta lahko omenimo primer sklicevanja na prebivalce v državi. Namesto pomnjenja (samo) osebnega imena, državni organi organizirajo svoje evidence glede na drugačen identifikator, kot sta npr. EMŠO ali davčna številka.

6.4 Imenska storitev DNS

Čeprav smo spoznali, da na omrežni plasti naprave naslavljamo z njihovim naslovom IP, človeški uporabniki le redkodaj uporabljamo takšno naslavljanje s številkami. Namesto pomnjenja omrežnih naslovov si veliko lažje zapomnimo besedilne ali *simbolične* naslove, ki jih za potrebe lažjega razumevanja lahko tudi smiselno hierarhično organiziramo. Na ta način si lažje zapomnimo www.google.com kot pa 172.217.19.100, ucilnica.fri.uni-lj.si namesto 212.235.188.23; istočasno pa lahko tudi iz naslova razberemo, da končnica uni-lj.si zaobjema celo hierarhijo članic univerze, kjer je FRI samo ena izmed njih. Jasno je, da je naslavljanje s simboličnimi imeni za uporabnika bolj preprosto in transparentno, saj lahko omrežne naslove po potrebi tudi spremenimo, pri tem pa z nekaj konfiguracije ohranimo isto simbolično ime.

Iz opisanega je skoraj samoumevno, da za naslavljanje omrežnih naprav z njihovimi simbolnimi imeni namesto z naslovi potrebujemo aplikacijsko storitev, ki zna za podani simbolični naslov poiskati pripadajoč omrežni naslov. Storitev, ki opravlja tovrstno nalogu, imenujemo DNS (angl. *Domain Name Service*). Poleg preslikovanja izbranega imena v pripadajoči naslov omogoča DNS tudi preslikovanje istega imena v različne omrežne naslove, s čimer se lahko zagotovi tudi porazdeljevanje bremena.

Storitev deluje z uporabo porazdeljene podatkovne zbirke, ki je organizirana hierarhično, in aplikacijskega protokola DNS za poizvedovanje po tej zbirki. Prednosti hierarhične organizacije zbirke DNS je več. Glavni razlog je, da se s tem izognemo enotni točki odpovedi in preobremenitvi oddaljene podatkovne baze, do katere bi sicer centralizirano dostopali vsi uporabniki interneta. Lahko si

tudi predstavljamo, da bi sistematično vzdrževanje in zagotavljanje nenehne razpoložljivosti takšne podatkovne baze bilo zelo težko. Ob razdelitvi podatkov in poizvedb na več omrežnih naprav zagotavljamo torej višjo *skalabilnost* (sposobnost odzivnega delovanja ne glede na večanje števila uporabnikov) in zanesljivost.

V hierarhiji DNS ločimo tri nivoje strežnikov:

1. **Korenski strežniki** (angl. *root servers*): so temeljni strežniki, ki so v DNS hierarhiji najvišje in skrbijo za usmerjanje uporabnika na strežnike v drugem nivoju hierarhije. Ker so ključni za delovanje storitve, so dostopni kar s 13 simboličnimi imeni (z imeni a.root-servers.net, ..., m.root-servers.net), od katerih vsako dejansko ne predstavlja samo enega strežnika, temveč celo gručo strežnikov, ki so v stanju medsebojne pripravljenosti v primeru izpada enega od njih. 13 omenjenih gruč strežnikov je tudi geografsko razprostrto po celotnem svetu, za njih pa skrbi 12 različnih organizacij.
2. **Strežniki za krovne domene (TLD)** (angl. *top level domain (TLD) servers*): so strežniki, ki skrbijo za krovne domene, kamor spadajo domene držav in generične domene. V letu 2017 je obstajalo 255 krovnih imen držav, kot so .si (Slovenija), .it (Italija), .rs (Srbija), .de (Nemčija), .tv (Tuvalu), .am (Armenija), .gl (Grenlandija). V zadnjih letih lahko opažamo prisotnost komercializacije teh domen, saj poddomene znotraj teh držav lahko prodajajo svoja imena kot priročne skovanke, ki omogočajo lažji dostop in uspešnejše trženje (npr. domene instagr.am, youtu.be, bi.ng, ti.me, pep.si, redd.it). Poleg imen držav obstajajo tudi t.i. generične krovne domene. Prvotnim sedmim generičnim domenam, ki so predstavljale vrsto organizacije (.com, .org, .net, .int, .edu, .gov in .mil), se je do leta pridružilo že približno 1500 dodatnih generičnih imen, ki še bolj podrobno opisujejo vrsto organizacije ali storitve (npr. .actor, .adult, .museum, .baseball, .blog, .news, .restaurant ipd.).
3. **Avtoritativni strežniki** (angl. *authoritative servers*): so strežniki, ki skrbijo za domene znotraj krovnih domen in po potrebi tudi za poddomene znotraj njih. Npr. avtoritativni strežnik Univerze v Ljubljani (za domeno uni-lj.si) hrani podatke o simboličnih imenih znotraj te domene (npr. za spletni strežnik www.uni-lj.si) in o avtoritativnih strežnikih za posamezne poddomene (npr. za fri.uni-lj.si, fmf.uni-lj.si ali fu.uni-lj.si).

Postopek poizvedovanja po omrežnem naslovu, ki pripada podanemu simboličnemu imenu, poteka kot zaporedje poizvedb po zgornji hierarhiji strežnikov. Denimo, da poizvedujemo po naslovu IP strežnika www.poddomena.domena.si. Prva poizvedba bo tedaj poslana enemu korenskemu strežniku za domeno .si. Sledila bo poizvedba,

poslana krovnemu (TLD) strežniku za domeno `.si`, z vprašanjem po avtoritativnem strežniku za domeno `domena.si`. Tretja v nizu bo poizvedba omenjenemu avtoritativnemu strežniku za domeno `domena.si` s poizvedbo po naslovu avtoritativnega strežnika za domeno `poddomena.domena.si`. Končno, s poizvedbo, poslano avtoritativnemu strežniku za domeno `poddomena.domena.si` z vprašanjem po naslovu računalnika `www.poddomena.domena.si`, bo ta strežnik vrnil želeni omrežni naslov.

6.4.1 Predpomnjenje zapisov DNS

V arhitekturi naprav, ki sodelujejo pri izvedbi storitev DNS poznamo tudi *lokalne strežnike DNS*. To so strežniki, ki jih običajno namestimo lokalno (v podjetju, pri ponudniku internetnih storitev) in igrajo vlogo posrednikov do hierarhije DNS. Namesto da svoje poizvedbe *iterativno* usmerjamo po hierarhiji strežnikov DNS (naprej korenskemu, nato krovnemu, nato avtoritativnim), lahko lokalnemu strežniku DNS posredujemo eno samo poizvedbo, ki jo imenujemo *rekurzivna poizvedba*, in mu prepustimo, da opravi nalogu iterativnega poizvedovanja namesto nas sam.

Uporaba rekurzivnih poizvedb nam prinaša dve pomembni prednosti. Prva je ta, da z njimi razbremenimo končne cliente pri poizvedovanju in jim omogočimo, da do rezultatov poizvedb pridejo z eno samo poslano poizvedbo. Druga pomembna prednost je ta, da lahko lokalni strežnik DNS hrani (predpomni, angl. *caching*) rezultate opravljenih poizvedb in namesto ponovnega poizvedovanja vrača shranjene odgovore. Poleg predpomnjenja odgovorov za celotna simbolična imena strežnikov, lahko strežniki DNS predpomnijo tudi delne rezultate poizvedb. Te vsebujejo naslove krovnih (TLD) strežnikov, s čimer razbremenimo korenske strežnike, in avtoritativnih strežnikov, s čimer razbremenimo krovne in druge avtoritativne strežnike. Z delnim predpomnjenjem lahko lokalni strežnik DNS skrajša verigo iterativnih poizvedb in nadaljuje od tiste točke v hierarhiji, za katero še nima predpomnjenih podatkov.

Pomembno je poudariti, da s tem dosežemo hitrejši odziv pri poizvedbah DNS in in manj prometa v omrežju. To je ključnega pomena zato, ker je DNS podpora storitev, ki jo posredno prožijo druge storitve (npr. zahtevek HTTP) in torej predstavlja del čakanja na izvedbo drugih storitev. Iz istega razloga se predpomnjenje zapisov DNS izvaja tudi na lokalnih računalnikih, s čimer se lahko v celoti izognemo izvedbi poizvedb preko omrežja.

6.4.2 Zapisi virov DNS

Iz dosedaj opisanega razumemo, kako je oblikovana hierarhija strežnikov DNS in da protokol deluje na principu odjemalec-strežnik. Omenili pa smo, da je poleg porazdeljenosti strežnikov porazdeljena tudi podatkovna baza vseh zapisov. Vsak

strežnik v hierarhiji vsebuje podatkovno bazo, v kateri se nahaja le del zapisov, za katere je odgovoren. Te zapise imenujemo *zapisi virov* (angl. *resource records, RR*), predstavljeni pa so s četverico podatkov: (tip, naziv, vrednost, TTL). V letu 2020 je pogosto uporabljenih vrst tipov zapisa okoli 45, ta seznam pa se posodablja z razvojem Interneta in potrebah ob rojevanju novih aplikacijskih protokolov¹⁰. Seznam najbolj pogosto uporabljenih tipov zapisov je prikazan v tabeli 6.9.

| tip zapisa | naziv | vrednost | pomen |
|------------|------------------------------|----------------------------------|---|
| A | simbolično ime | naslov IPv4 | zapis naslova (address), ki hrani simbolično ime in IP številko; ti zapis so običajno shranjeni v avtoritativnih strežnikih za izbrano domeno |
| AAAA | simbolično ime | naslov IPv6 | predstavlja naslov IPv6 (analogno tipu A) |
| NS | ime domene | ime avtoritativnega strežnika | ti zapis običajno v TLD strežnikih za iskanje avtoritativnega strežnika neke domene |
| CNAME | psevdonim (nadomestno ime) | kanonično (pravo) ime | omogoča bolj prijazno poimenovanje javno dostopnih strežnikov (npr. www.ibm.com je dejansko www.ibm.com.cs186.net) |
| MX | psevdonim poštnega strežnika | kanonično ime poštnega strežnika | omogoča bolj prijazno poimenovanje domenskega poštnega strežnika (tako lahko npr. uporabnika s poštnim predalom na strežniku mta5.am0.yahoodns.net lažje naslovimo z ime@yahoo.com) |
| PTR | naslov IP | simbolično ime | zapis, ki omogoča vzvratne DNS poizvedbe (poizvedovanje po imenu na podlagu naslova IP) |
| CERT | enkripcijski algoritem | certifikat | hrani certifikate za kriptiranje (npr. PKIX, SPKI, PGP). |

Tabela 6.9 Seznam pogostih tipov zapisov v strežnikih DNS

6.4.3 Protokol DNS

Poleg hierarhije strežnikov in porazdeljene podatkovne zbirke je tretji kos sestavljanke pri storitvi DNS protokol za poizvedovanje po zbirki. Protokol DNS deluje po principu izziv-odgovor, strežnik pa sprejema zahtevke na vratih UDP

¹⁰Definirani v številnih dokumentih RFC, kot so: 1035, 1183, 1876, 2230, 2535, 2782, 2845, 2930, 3123, 3403, 3596, 4025, 4034, 4255, 4398, 4431, 4701, 5155, 6672, 6698, 6844, 7344, 7477, 7553, 7929, 8005, 8162, 8482.

★ **Primer 6.7:** Denimo, da odpremo novo podjetje in registriramo domeno `mojafirma.com` pri lokalnem registrarju. Kaj je potrebno storiti, da lahko v svoji domeni usposobimo delovanje storitve DNS za domenski spletni in poštni strežnik?

Ob postaviti lokalnega omrežja moramo postaviti domači avtoritativni strežnik, ki bo skrbel za preslikave imen strežnikov znotraj naše domene. Denimo, da ta strežnik pojmenujemo `dns1.mojafirma.com`. Ime in naslov tega avtoritativnega strežnika sporočimo registrarju, ki dopolni bazo ustreznega krovnega strežnika (TLD) z zapisoma, ki hranita podatke o našem avtoritativnem strežniku:

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

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

V lokalni avtoritativni strežnik lahko tedaj vnesemo zapise tipa A (AAAA) za spletni in ostale javne strežnike. Če želimo olajšati uporabnikom Interneta dostop do spletnega strežnika, lahko ustrezno dodamo tudi zapise tipov CNAME in MX.

53. Strežnik ne hrani stanja povezav, v primeru izgube aplikacijskih podatkov pa na aplikacijski ravni skrbi za ponovno pošiljanje.

Zahlevki in odgovori aplikacijskih sporočil imajo enako strukturo. Ta je prikazana na sliki 6.6. S slike lahko razberemo, da sporočilo vsebuje naslednje poglavitne komponente:

- **identifikator** (16 bitov): polje, ki povezuje zahtevke z odgovori,
- **zastavice in status** (16 bitov): zahteva/odgovor, vrsta poizvedbe, avtoritativnost odgovora, krajanje sporočila, razpoložljivost rekurzije, zahtevanje rekurzije, status odgovora,
- **število poizvedb**, ki so zaobjete v zahtevku,
- **število zapisov**, ki so vključeni v **polju z odgovorom**,
- **število zapisov** v polju za **avtoritativne odgovore** strežnika,
- **število dodatnih** pomožnih zapisov v temu namenjenem polju.

Zgornja struktura kaže, da z enim samim zahtevkom lahko strežniku posredujemo več zahtevkov in prejmemo več odgovorov. Primer ročno izvedene poizvedbe z orodjem `nslookup` je prikazan na primeru 6.8.

~~~~~ **Varnostni pomislek:** Pozoren bralec lahko hitro pomisli na to, da lahko s spremenjanjem zapisov v strežnikih DNS preusmerjamo, kam bo poslan promet, ki je namenjen napravi z nekim simboličnim namenom. To lahko izkoristijo napadalci, ki s postopkom **zastrupljanja tabel DNS** (angl. *DNS poisoning*) poskrbijo za napačne vnose, na podlagi katerih omrežni promet steče proti njihovim strežnikom. Na teh lažnih naslovih strežnikov napadalci radi postavijo spletišča, ki so na videz skoraj identična originalnim, njihov namen pa je zavajanje uporabnikov v posredovanje nakazil, gesel ali drugih osebnih podatkov. Podobne učinke preusmerjanja prometa lahko izvedemo tudi na nižjih plasteh

| identifikator                             | zastavice                                                |                                                               |
|-------------------------------------------|----------------------------------------------------------|---------------------------------------------------------------|
| število poizvedb                          | število zapisov v odgovoru                               | 12 bajtov                                                     |
| število avtoritativnih zapisov v odgovoru | število dodatnih zapisov v odgovoru                      |                                                               |
|                                           | poizvedbe<br>(spremenljivo število)                      | parametri poizvedbe:<br>URL, tip, razred                      |
|                                           | odgovori<br>(spremenljivo število zapisov)               |                                                               |
|                                           | avtoritativni odgovori<br>(spremenljivo število zapisov) | deli odgovorov:<br>URL, tip, razred,<br>TTL, dolžina, odgovor |
|                                           | dodatni zapisi<br>(spremenljivo število zapisov)         |                                                               |

Slika 6.6 Aplikacijsko sporočilo protokola DNS

komunikacijskega modela, npr. z zastrupljanjem stikalnih tabel ali tabel ARP. Za odpravljanje opisane nevarnosti pri protokolu DNS lahko uporabimo razširitev DNSSEC (angl. *DNS Security Extensions*), ki omogoča digitalno podpisovanje zapisov, za katere je strežnik avtoritativen.

## 6.5 Protokoli P2P

Poleg protokolov, ki delujejo na principu odjemalec-strežnik poznamo tudi protokole, ki se izvajajo neposredno med uporabniki (angl. *peer-to-peer, P2P*). Arhitektura omrežnih aplikacij je pri teh protokolih drugačna, saj ne zahtevajo strežnika, ki bi moral biti nenehno prižgan. Odjemalci lahko k uporabi takšne omrežne aplikacije pristopajo po potrebi in ravno tako iz nje tudi odhajajo. Na ta način morajo protokoli iz družine P2P biti sposobni zagotoviti izmenjavo podatkov med poljubnima končnima sistemoma.

Izstopajoča prednost pri uporabi aplikacij P2P je njihova *skalabilnost*, t.j. sposobnost odzivnega delovanja ne glede na večanje števila uporabnikov. Ta je zagotovljena ravno s pristopom, da je omogočena izmenjava podatkov med poljubnim parom odjemalcev. Namesto da bi čas dostave neke datoteke vsem odjemalcem naraščal linearno s številom uporabnikom, kot je res pri arhitekturi odjemalec-strežnik, čas dostave pri protokolih P2P zato narašča veliko počasneje in bolj podobno logaritemski funkciji. Aplikacije P2P imajo tudi izzive, saj je potrebno zagotoviti njihovo delovanje tudi, če uporabniki po priklopu menjajo svoj IP naslov in se nahajajo v zasebnih omrežjih za usmerjevalniki, ki izvajajo preslikovanje naslovov (NAT). Izmenjava podatkov s poljubnim neznanim uporabnikom ravno tako odpira varnostna vprašanja, pri deljenju datotek pa je

★ **Primer 6.8:** Primer poizvedbe DNS z orodjem nslookup. Lokalnemu strežniku zastavimo vprašanje po omrežnem naslovu strežnika `www.abc.com`. Izpis prikazuje odgovor, ki vsebuje zapise tipov A, AAAA in CNAME.

```

> set debug
> set d2
> set recurse
> www.abc.com
Server: UnKnown
Address: 192.168.1.254

-----
SendRequest(), len 29
HEADER:
opcode = QUERY, id = 2, rcode = NOERROR
header flags: query, want recursion
questions = 1, answers = 0, authority records = 0, additional = 0
QUESTIONS:
www.abc.com, type = A, class = IN
-----

Got answer (136 bytes):
HEADER:
opcode = QUERY, id = 2, rcode = NOERROR
header flags: response, want recursion, recursion avail.
questions = 1, answers = 5, authority records = 0, additional = 0
QUESTIONS:
www.abc.com, type = A, class = IN
ANSWERS:
-> www.abc.com
type = CNAME, class = IN, dlen = 31
canonical name = d2iwww1xxkqpmiz.cloudfront.net
ttl = 300 (5 mins)
-> d2iwww1xxkqpmiz.cloudfront.net
type = A, class = IN, dlen = 4
internet address = 99.86.243.39
ttl = 60 (1 min)
<izpis skrajšan>

-----
Non-authoritative answer:
SendRequest(), len 29
HEADER:
opcode = QUERY, id = 3, rcode = NOERROR
header flags: query, want recursion
questions = 1, answers = 0, authority records = 0, additional = 0
QUESTIONS:
www.abc.com, type = AAAA, class = IN
-----

Got answer (296 bytes):
HEADER:
opcode = QUERY, id = 3, rcode = NOERROR
header flags: response, want recursion, recursion avail.
questions = 1, answers = 9, authority records = 0, additional = 0
QUESTIONS:
www.abc.com, type = AAAA, class = IN
ANSWERS:
-> www.abc.com
type = CNAME, class = IN, dlen = 31
canonical name = d2iwww1xxkqpmiz.cloudfront.net
ttl = 300 (5 mins)
-> d2iwww1xxkqpmiz.cloudfront.net
type = AAAA, class = IN, dlen = 16
AAAA IPv6 address = 2600:9000:206e:f600:a:896e:12c0:93a1
ttl = 60 (1 min)
-> d2iwww1xxkqpmiz.cloudfront.net
type = AAAA, class = IN, dlen = 16
AAAA IPv6 address = 2600:9000:206e:a600:a:896e:12c0:93a1
ttl = 60 (1 min)
<izpis skrajšan>

-----
Name: d2iwww1xxkqpmiz.cloudfront.net
Addresses: 2600:9000:206e:f600:a:896e:12c0:93a1
2600:9000:206e:a600:a:896e:12c0:93a1
2600:9000:206e:d000:a:896e:12c0:93a1
2600:9000:206e:8c00:a:896e:12c0:93a1
2600:9000:206e:7e00:a:896e:12c0:93a1
2600:9000:206e:fe00:a:896e:12c0:93a1
2600:9000:206e:9800:a:896e:12c0:93a1
2600:9000:206e:2a00:a:896e:12c0:93a1
99.86.243.39
99.86.243.109
99.86.243.71
99.86.243.75
Aliases: www.abc.com

```

potrebno skrbeti tudi za motiviranje uporabnikov, da prispevajo ostalim v enakem pravičnem deležu podatkov, kot so jih tudi sami prejeli od drugih.

### 6.5.1 BitTorrent

Danes verjetno najbolj znan protokol P2P za deljenje datotek je protokol BitTorrent. Izmenjava datotek pri uporabi tega protokola poteka med gručo uporabnikov, med katerimi se prenašajo manjši kosi deljene datoteke, imenovani koščki (angl. *chunks*). Gruči uporabnikov, med katerimi poteka izmenjava koščkov datoteke, se lahko novi uporabniki pridružijo ob poljubnem času. Pridružitev novega uporabnika poteka tako, da se ta prijavi sledilnemu strežniku (angl. *tracker*), od katerega dobi seznam ostalih odjemalcev, ki tvorijo gručo za izmenjavo želene datoteke. Ker je seznam vseh odjemalcev v gruči lahko zelo velik in hkratno komuniciranje z njimi vpliva na performanse, odjemalec izbere le podmnožico vseh uporabnikov, imenovanih sosed (angl. *neighbors*). Nadaljnja izmenjava koščkov datotek poteka le s temi sosedji.

Poleg podatkov, vsebujejo aplikacijska sporočila tudi dva kontrolna bita, ki opisujeta stanje povezave med odjemalcema: zamašen (angl. *choked*) in zainteresiran (angl. *interested*). Vsaka povezava je na začetku inicializirana v *zamašenem* in *nezainteresiranem* stanju, pretok podatkov pa se začne izvajati takrat, kadar je stanje prvega odjemalca *zainteresiran*, drugi odjemalec pa ni nastavil stanja *zamašen*, s čimer je razglasil, da ne bo odgovarjal na zahteve prvega. Nastavljanje statusa *zamašen* je pri protokolu BitTorrent koristno, ker za odjemanje datotek potrebujemo več različnih TCP povezav. V tem primeru nadzor zasičenja, ki je vgrajen v TCP, deluje ločeno in neuskajeno za vsako od posameznih povezav. Z uvedbo dodatnega nadzora zasičenja na aplikacijski plasti protokol BitTorrent to težavo odpravi.

Pri odjemanju razpoložljivih koščkov datotek lahko odjemalec pristopa k njihovemu prenosu z različnimi strategijami. Prenosa se sicer lahko loti v naključnem vrstnem redu, a bolj varna strategija je, da se loti najprej prenosa koščkov, ki so med sosedji najmanj prisotni (angl. *rarest first*). Na ta način si odjemalec lahko zagotovi bolj verjeten uspešen prenos celotne datoteke ob nevarnosti, da uporabniki z redkimi koščki predčasno zapustijo gručo. Med prejemanjem koščkov datotek je naloga odjemalca tudi, da svoje koščke deli z ostalimi uporabniki na enak način, kot je pridobil svoje koščke. Da protokol poskrbi za vzajemnost sodelovanja, odjemalec svoje koščke pošilja s hitrostjo, ki je sorazmerna hitrosti, s katero od njih koščke prejema.

Cilj vsakega odjemalca je torej, da sčasoma pridobi vse manjkajoče koščke datoteke. Ob zaključku prenosa k sebi lahko odjemalec še nekaj časa ostane v gruči in deli koščke z drugimi uporabniki, lahko pa tudi predčasno zapusti gručo in s tem morda odnese nerecipročno več podatkov, kot jih je poslal drugim. Iz tega razloga upravljavci nekaterih sledilnikov nadzorujejo razmerja med prejetim

in poslanim prometom posameznih uporabnikov in od uporabnikov zahtevajo vrednosti teh razmerij, ki so še sprejemljiva.

### 6.5.2 Skype

Tudi aplikacija Skype spada med najbolj znane aplikacije za telekonference, t.j. za prenos videa in govora. Aplikacija uporablja za komunikacijo *lastniški* protokol, katerega specifikacija pa ni javno objavljena. S skrivanjem specifikacije podjetje skriva svoje finančne interese, saj si z učinkovitim delovanjem aplikacije zagotavljajo konkurenčno prednost na trgu. Ne glede na zapisano pa so podrobnosti o delovanju protokola Skype že uspešno razvozljali uporabniki s postopkom *obratnega inženiringa*, ki jih je ob opazovanju omrežnega prometa pripeljal do smiselnih zaključkov o delovanju.

Ob zagonu aplikacije se uporabnik najprej poveže s strežnikom za prijavo, ki hrani uporabniška imena in gesla uporabnikov. Po uspešni avtentikaciji se odjemalec nato poveže do najbližjega **nadzornega vozlišča** (angl. *supernodes*), ki na nek način predstavlja vstopno točko v aplikacijsko "hrbtenico" aplikacije Skype. Nadzorna vozlišča imajo dve bistveni nalogi:

- hranijo zapise, ki povezujejo **uporabniška imena** uporabnikov z njihovimi **omrežnimi naslovi IP**,
- skrbijo za **vzpostavljanje povezave** med odjemalci.

Ker mora Skype zagotavljati povezljivost med poljubnimi uporabniki, se pri vzpostavitvi videokonferenčne zveze do uporabnika, ki je skrit za mehanizmom NAT, pojavijo težave. Te Skype rešuje na naslednji izviren in zanimiv način:

1. Udeleženec A, ki želi vzpostaviti zvezo, obvesti o svojem interesu svoje nadzorno vozlišče NA. NA ta interes posreduje nadzornemu vozlišču NB ciljnega uporabnika B.
2. Ob posredovanju interesa se v omrežju Skype izbere nadzorno vozlišče NR, ki bo igralo vlogo posrednika/premostitvenega vozlišča (angl. *relay*) pri povezavi.
3. Uporabnika NA in NB se obvesti o naslovu posrednika NR in ju s tem povabi k vzpostavitvi povezave z njim. Ker sta oba uporabnika vzpostavila odhodno povezavo skozi svoj usmerjevalnik in s tem ta shrani njune naslove v tabeli NAT, lahko komunikacija med uporabnikoma nemoteno steče.
4. Premostitveno vozlišče v nadaljevanju obvesti uporabnika A o javnem naslovu/vratih uporabnika B, preko katerem uspešno komunicira, in uporabnika B o naslovu/vratih uporabnika A. A in B lahko začneta od te točke naprej komunicirati neposredno, mimo premostitvenega vozlišča.



## 7 Kriptografija in omrežna varnost

Ob nastanku je bil Internet namenjen peščici raziskovalcev in podjetij, ki so omrežja uporabljala zlasti za izmenjavo datotek, za pošiljanje elektronske pošte in za deljenje omrežnih virov, kot so tiskalniki. S kasnejšo hitro rastjo zmogljivosti in s širitevijo informacijske tehnologije je to globalno omrežje začelo uporabljati več milijonov uporabnikov za raznovrstne potrebe, kot so spletni nakupi, bančništvo, oddaja dohodninskih napovedi ipd. Postopki, ki jih sicer opravljamo v vsakodnevni življenju osebno, so se na ta način prenesli v obliko njihove elektronske izvedbe, kjer pa kot udeleženci v komunikaciji nismo zastopani kot fizične osebe, temveč se skrivamo za omrežnim naslovom.

Problem dokazovanja identitete, na katerega smo nakazali v prejšnjem odstavku, je le ena izmed varnostnih težav, ki jo srečamo v računalniških omrežjih. Razvoj omrežne programske opreme uvaja še dodatna druga tveganja, saj ima vsaka programska oprema lahko varnostne pomanjkljivosti, ki lahko posamezniku, ki uporablja omrežje, predstavljajo grožnjo. V računalniških omrežjih deluje tudi veliko število uporabnikov, ki jih bomo imenovali **napadalci**. To so uporabniki, ki radi izkoriščajo omenjene grožnje z namenom, da bi si pridobili kakšno korist (npr. finančno, informacijsko), da bi pritegnili pozornost (za slavo?) ali pa ciljno škodovali specifičnim osebam. Ti uporabniki izvajajo škodljive aktivnosti, kot so **prisluškovanje** komunikaciji (angl. *eavesdropping*), **ponarejanje** sporočil (angl. *message forging, spoofing*), **kraja identitete** (angl. *identity theft, impersonation*), prevzem aktivne seje ali **izločitev** udeleženca iz komunikacije (angl. *session hijacking*) ali **onemogočanje rabe storitev** (angl. *Denial of Service - DOS*). Ti uporabniki so lahko hekerji, jezni odpuščeni zaposleni, spletni prevaranti, vsiljive marketinške agencije, teroristi, skrivne vladne organizacije, (morda) študenti (☺) in drugi.

V tem poglavju bomo preučili pojem *omrežne varnosti* z različnih zornih kotov in pojasnili pristope, kako se lahko izognemo nevarnostim v omrežju in poskrbimo za varno komunikacijo.

## 7.1 Elementi omrežne varnosti

V splošnem lahko rečemo, da so glavni cilji omrežne varnosti dodati v omrežno komunikacijo vsaj nekatere izmed naslednjih elementov:

1. **Zaupnost** ("*Kdo lahko sliši?*"): poskrbeti, da tretje osebe, katerim komunikacijski dialog ni namenjen, ne morejo prisluškovati pogovoru.
2. **Integriteto** ("*Je bilo spremenjeno?*"): onemogočiti možnost, da lahko ne-pooblaščena oseba spreminja vsebino sporočil med udeležencema, ne da bi slednji lahko to zaznali, ali da bi ponovila shranjena stara sporočila (angl. communication replay).
3. **Identifikacijo** ("*Kdo si?*"): ugotavljanje identitete sogovornika.
4. **Avtentifikacijo** ("*Dokaži, da si to res ti*"): zmožnost dokazovanje lastne identitete ali identitete sogovornika. Avtentifikacija je pomemben element v varni komunikaciji, saj se na njeni podlagi lahko prepričamo o identiteti sogovornika, preden mu pošljemo občutljive podatke.
5. **Avtorizacijo** ("*Kaj lahko počnem?*"): poskrbeti, da lahko vsaka oseba dostopa samo do tistih virov, ki so ji namenjeni in do katerih je upravičena.
6. **Preprečevanje zanikanja komunikacije** ("*Si res prejel, si res poslal?*"): omogočanje dokazovanja, da je določen pošiljatelj res bil tisti, ki je sporočilo poslal. Vključuje tudi preprečevanje zanikanja (angl. nonrepudiation), da je nekdo res prejel ali poslal določeno sporočilo.

Vsi našteti problemi so prisotni tudi v vsakdanjem življenju, vendar so s fizičnim in osebnim stikom veliko lažje rešljivi kot v svetu elektronskih komunikacij. Za reševanje zaupnosti komunikacije se namreč zagotovo lahko premaknemo v prostor, kjer smo s sogovornikom sami, pomanjkanje integritete dokumenta lahko tudi zlahka zaznamo, če nam nekdo odda slabo fotokopijo spričevala, sogovornika lahko avtenticiramo po njegovem videzu, ravno tako lahko poskrbimo tudi za potrdila o izvedbi dejanj.

V svetu računalniških komunikacij rešujemo naštete probleme večplastno. Če se spomnimo komunikacijskega modela, ki smo ga omenili v uvodu knjige, in pomislimo, na katero plast bi varnost najbolj spadala, lahko hitro ugotovimo, da lahko pravzaprav doprinese prav na vseh plasteh komunikacijskega modela:

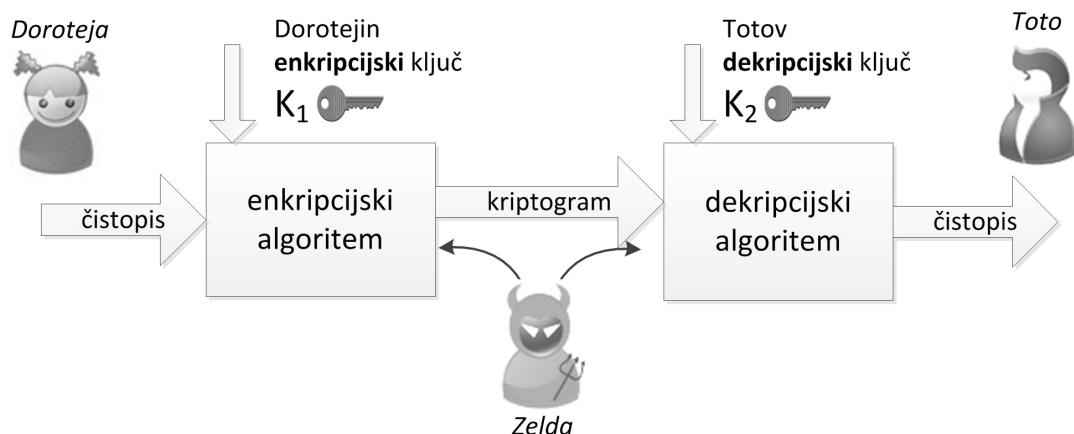
- na **fizični plasti** lahko zavarujemo povezavo s fizičnim varovanjem opreme, na povezavni plasti pa že s kriptiranjem (šifriranjem/zakrivanjem) vsebine okvirjev,
- na **omrežni plasti** lahko uporabimo posebne filtre paketov (požarni zidovi) in programsko opremo za opazovanje škodljivega prometa (sistemi za zaznavanje vodorov),
- na **transportni plasti** lahko poskrbimo za kriptiranje celotnih povezav med dvema končnima procesoma,

- na **aplikacijski plasti**, kjer so prisotna uporabniška imena, lahko skrbimo za avtentikacijo in preprečevanje zanikanja komunikacije.

Pri navajanju zgornjih rešitev se osredotočamo le na *tehnološki vidik* zagotavljanja varnosti v sistemu, ki pa ni edini. Pomemben je tudi *organizacijski vidik*, ki skrbi tudi za druge poglede varnosti, ki morajo biti prisotni, da je sistem vsestransko varen. Takšni vidiki so npr. fizično varovanje računalniške opreme (fizični dostop do nje, biometrična in drugačna avtentikacija itd.), izobraževanje uporabnikov, varnostna politika cele organizacije in uporabnikov (gesla, protivirusna zaščita, varnost računalnikov). Za tehnološki vidik varnosti, ki mu bomo posvetili največ pozornosti, lahko zatrdimo, da na vseh plasteh omrežnega sistema razen na fizični plasti uporablja **kiptografske pristope**.

**Kriptografija** je veda, katere ime izhaja iz grškega jezika in pomeni "skriti zapis" ali "zakrito sporočilo". Z besedo *zakrito* je seveda mišljeno, da sporočilo ni vidno očem nepovabljenih bralcev. V računalništvu je kriptografija veliko več kot le umetnost skrivanja (ali pa kriptiranja, šifriranja) vsebine sporočila. Je veda, katere korenine segajo globoko v matematiko in na podlagi katere so poleg zagotavljanja zaupnosti implementirane tudi metode za zagotavljanje integritete, avtentikacije in preprečevanja zanikanja komunikacije. V naslednjih razdelkih si bomo pogledali osnovne kriptografske pristope, ki nam bodo omogočili razumevanje konceptov posameznih pristopov varovanja.

**Komunikacijski model.** Razložimo najprej osnovno terminologijo, ki jo bomo uporabljali. Denimo, da bomo pri našem študiju preučevali varnost med dvema udeležencema v omrežni komunikaciji, ki ju bomo imenovali kar Doroteja in Toto. V našem svetu bosta ta dva uporabnika izvajala komunikacijo preko omrežja, v katerem se skriva tudi napadalca Zelda, ki jima želi škodovati (slika 7.1).



**Slika 7.1** Model komunikacije, ki vsebuje pošiljatelja, prejemnika in napadalca

Sporočilu, ki ga želi Doroteja poslati Totu in je v berljivem jeziku, bomo rekli **čistopis**. Če Doroteja pripravljen čistopis pošlje preko omrežja, v katerem prisluškuje Zelda, bo tudi Zelda lahko razbrala, o čem teče beseda, česar pa si morda ne želimo. Zaradi tega bo Doroteja svoj čistopis pred pošiljanjem spremenila v zakrito obliko tega besedila, ki jo bomo imenovali **kriptogram**. Za pretvorbo čistopisa v kriptogram bo Doroteja uporabila enega izmed znanih postopkov za zakrivanje sporočil, imenovanega **enkripcijski algoritem**, ki si ga je v zgodovini izmislil kak znani cesar, vojak, filozof ali matematik in je pokazal, da je ta postopek praktičen ali uporaben. Ker takšnih postopkov na svetu ni veliko, bi jih ob vsakodnevni kriptiranju kmalu izčrpali in s tem povečali verjetnost tega, da bo Zelda ob naključnem ugibanju ugotovila, kako je bil kriptogram tvorjen. Zato pri uporabi vsakega enkripcijskega algoritma poskrbimo, da bo ta deloval rahlo drugače, kar dosežemo s posebno nastavljivo tega algoritma (parametrom), ki ga imenujemo **enkripcijski ključ**. Za izdelavo kriptograma bo Doroteja torej uporabila nek enkripcijski ključ  $K_1$ , od katerega bo odvisno to, kako bo deloval enkripcijski algoritem. Ob prejemu Dorotejinega kriptograma ga bo Toto dekriptiral le, če pozna ustrezni dekripcijski ključ  $K_2$ . Z dekriptiranjem bo tako pridobil prvotni čistopis in prebral vsebino Dorotejinega sporočila.

Glede na enakost/neenakost ključev  $K_1$  in  $K_2$  ločimo dve veliki družini kriptografskih metod, simetrične in nesimetrične. **Simetrične** metode so tiste, kjer sta enkripcijski in dekripcijski ključ enaka (torej,  $K_1 = K_2$ ), z njimi bomo tudi začeli naš pregled kriptografskih metod. Nesimetrične metode ali *kriptografija z javnimi ključi* so drugačne, saj uporabljajo par ključev, ki sta si različna (torej,  $K_1 \neq K_2$ ). Varnost kriptografskih metod vrednotimo na podlagi tega, kako lahko morebitni napadalec izvede **kriptoanalizo**, t.j. da ugotovi, kako *razbiti* kriptografsko metodo in ugotoviti, kakšen čistopis se skriva za kriptogrami.

Pri našem kasnejšem študiju mehanizmov za zagotavljanje integritete besedila bomo spoznali tudi še eno, posebno vrsto kriptografskih algoritmov, imenovanih **zgoščevalne funkcije** (angl. *hash functions*), ki ne uporabljajo ključev, čistopis pa nepovratno kriptirajo v kriptogram, katerega ni možno prevesti nazaj v čistopis. Pa je to sploh uporabno? Seveda! Vendar ne prehitevajmo, vse ob svojem času.

## 7.2 Preprosti kriptografski pristopi

V zgodovini so kriptografske metode delili v dve veliki kategoriji: **substitucijske (ali zamenjalne) metode** (zamenjava znakov za druge znake) in **transpozicijske (ali izmenjalne) metode** (zamenjava vrstnega reda znakov v čistopisu). Čeprav sodobnih kriptografskih metod ne moremo odločno uvrstiti le v eno od naštetih kategorij, si bomo vendarle za razumevanje osnovnih principov kriptografije najprej pogledali najbolj preproste izmed naštetih. Poglavlje bomo zaključili s

★ **Primer 7.1:** Za primer si poglejmo Cezarjevo kriptiranje, kjer izberemo ključ  $k=6$ . Pomoč pri izbiri znakov za substitucijo je prikazana v tabeli 7.1 (substitucijska tabela). Denimo, da želimo kriptirati čistopis **LEGIJA**. S pomočjo spodnje tabele ga bomo torej preslikali v kriptogram **SKMOPF**.

pregledom sodobnejših kriptografskih pristopov.

### 7.2.1 Substitucijske metode

Princip delovanja substitucijskih metod je, da vsak znak (ali skupino znakov) nadomestijo z znakom (ali skupino znakov) in s tem čistopis prevedejo v kriptogram.

**Cesarjev kriptogram.** Ena najstarejših tovrstnih metod se imenuje po vladarju Juliju Cezarju, ki je substitucijsko kriptiranje uporabljal za zakrivanje vojaških sporočil. Za kriptiranje po izvornem Cezarjevem postopku postopamo tako, da vsako posamezno črko čistopisa zamaknemo po abecedi<sup>1</sup> naprej za tri znake; če pri zamikanju pridemo do konca abecede, začnemo ponovno od njenega začetka. Na ta način *a* postane *č*, *b* postane *d*, ..., *ž* pa postane *c*. Omenjeno Cezarjevo metodo lahko posplošimo na ta način, da si pri konkretnem kriptiranju izberemo, za koliko črk zamikamo neko določeno sporočilo. Če označimo torej naš izbrani zamik s *k*, to pomeni, da igra naš *k* dejansko vlogo kriptografskega ključa (parametra, ki je potreben za kriptiranje in dekriptiranje sporočila - glej razdelek 7.1).

|            |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |
|------------|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| čistopis   | A | B | C | Č | D | E | F | G | H | I | J | K | L | M | N | O | P | R | S | Š | T | U | V | Z | Ž |
| kriptogram | F | G | H | I | J | K | L | M | N | O | P | R | S | Š | T | U | V | Z | Ž | A | B | C | Č | D | E |

**Tabela 7.1** Substitucijska tabela za Cezarjev kriptogram

Hitro lahko razberemo, da je opisana metoda dokaj preprosta in da bi morebitni napadalec ali kriptoanalitik lahko metodo razvozlal v 25 poskusih – s poskušanjem, kateri od 25 možnih zamikov abecede prevede kriptogram nazaj v berljiv čistopis. S tem namenom lahko opisano metodo še bolj posplošimo tako, da prvotno abecedo preslikamo v poljubno permutacijo črk abecede, pri čemer ni potrebno, da zaporedne črke ohranjajo abecedno urejenost. Na ta način bi moral morebitni napadalec v splošnem opraviti  $25!$  (25 fakulteta) poskusov<sup>2</sup>, da bi prišel

<sup>1</sup>Pri kriptogramih, ki uporablajo abecedo, se je potrebno odločiti, katero abecedo bomo uporabljali. Za potrebe učbenika se odločimo, naj bo to slovenska abeceda, ki jo tvori 25 črk.

<sup>2</sup>V praksi se izkaže, da potrebuje veliko manj poskusov, če upošteva značilnosti jezika in frekvence posameznih zlogov.

★ **Primer 7.2:** Za primer si izberimo, da ima naš kriptografski ključ vrednost PAMET, kriptirati pa želimo čistopis SKRIVNOST, kot je prikazano v tabeli 7.2. Zasenčene vrstice v tabeli 7.3 torej kažejo, katere abecede bomo uporabili za kriptiranje. Znotraj vsake besede bomo za izbrano črko čistopisa poiskali črko v tej abecedi (te so v tabeli 7.3 senčene temneje), s čimer bomo dobili kriptogram in čistopis kriptirali tako, da dobimo kriptogram IKENREOFA.

do čistopisa. Kadar uporabljamo kriptiranje z eno samo abecedo, imenujemo takšno metodo **enoabecedna (monoalfabetska) substitucija** (angl. *monoalphabetic substitution*).

**Vigenèrjev kriptogram.** Je substitucijski kriptogram, ki izvaja večabecedno (polialfabetsko) substitucijo. Vigenèrjevo kriptiranje izvajamo tako, da izberemo ključ, ki je predstavljen z neko besedo, in vsako zaporedno črko čistopisa kriptiramo z naslednjo abecedo, ki jo določa soležna črka ključa. V primeru, če je črk v ključu premalo, črke ključa uporabimo krožno, znova od začetka. Če ima ključ  $k$  torej  $len(k)$  znakov, to pomeni, da z  $i$ -to črko ključa kriptiramo črke čistopisa na mestih  $i, i + len(k), i + 2 \cdot len(k)$  itd.

|                 |   |   |   |   |   |   |   |   |   |
|-----------------|---|---|---|---|---|---|---|---|---|
| čistopis        | S | K | R | I | V | N | O | S | T |
| ključ (abeceda) | P | A | M | E | T | P | A | M | E |
| kriptogram      | I | K | E | N | R | E | O | F | A |

Tabela 7.2 Kriptiranje z Vigenèrjevim kriptogramom

Ker je kompleksnost kriptograma odvisna od dolžine in vsebine ključa, hitro vidimo, da je takšen kriptogram v primerjavi s Cesarjevim bolj odporen na statistično analizo parov in trojic črk.

**Porterjev kriptogram.** Porterjev kriptogram je posebna oblika Vigenèrjevega kriptograma, ki kriptira pare znakov (bigrame) hkrati. Na ta način lahko znaka AB npr. kriptira v PH, znaka AC v RWXX (substitucijski elementi so lahko zaporedja črk različnih dolžin, simboli, barve, karkoli) ipd. Ključ takšnega kriptograma je torej predstavljen s celotno matriko (ali pa seznamom) preslikav vseh možnih parov znakov. Iz Porterjevega kriptograma lahko vidimo, da lahko v splošnem izvaja tudi kompresijo (stiskanje) kriptogramov, saj nadomesti več znakov čistopisa z enim samim.

**Kodiranje.** Med substitucijske kriptograme spada tudi metoda kodiranja. Gre za postopek, pri katerem posamezne znake, besede ali pa celo sporočilo nadomestimo z dogovorjenim substitucijskim simbolom. Ključ za kodiranje je

|  |  | črka čistopisa                      |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |  |
|--|--|-------------------------------------|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|--|
|  |  | izbira abecede glede na črko ključa |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |  |
|  |  | A                                   | B | C | Č | D | E | F | G | H | I | J | K | L | M | N | O | P | R | S | Š | T | U | V | Z | Ž |  |
|  |  | B                                   | C | Č | D | E | F | G | H | I | J | K | L | M | N | O | P | R | S | Š | T | U | V | Z | Ž | A |  |
|  |  | C                                   | C | D | E | F | G | H | I | J | K | L | M | N | O | P | R | S | Š | T | U | V | Z | Ž | A |   |  |
|  |  | C                                   | Č | D | E | F | G | H | I | J | K | L | M | N | O | P | R | S | Š | T | U | V | Z | Ž | A |   |  |
|  |  | C                                   | D | E | F | G | H | I | J | K | L | M | N | O | P | R | S | Š | T | U | V | Z | Ž | A | B |   |  |
|  |  | D                                   | E | F | G | H | I | J | K | L | M | N | O | P | R | S | Š | T | U | V | Z | Ž | A | B | C |   |  |
|  |  | E                                   | F | G | H | I | J | K | L | M | N | O | P | R | S | Š | T | U | V | Z | Ž | A | B | C | Č |   |  |
|  |  | F                                   | G | H | I | J | K | L | M | N | O | P | R | S | Š | T | U | V | Z | Ž | A | B | C | C | D |   |  |
|  |  | G                                   | H | I | J | K | L | M | N | O | P | R | S | Š | T | U | V | Z | Ž | A | B | C | C | D | E |   |  |
|  |  | H                                   | I | J | K | L | M | N | O | P | R | S | Š | T | U | V | Z | Ž | A | B | C | C | D | E | F |   |  |
|  |  | I                                   | J | K | L | M | N | O | P | R | S | Š | T | U | V | Z | Ž | A | B | C | C | D | E | F | G |   |  |
|  |  | J                                   | K | L | M | N | O | P | R | S | Š | T | U | V | Z | Ž | A | B | C | C | D | E | F | G | H |   |  |
|  |  | K                                   | L | M | N | O | P | R | S | Š | T | U | V | Z | Ž | A | B | C | C | D | E | F | G | H | I |   |  |
|  |  | L                                   | M | N | O | P | R | S | Š | T | U | V | Z | Ž | A | B | C | C | D | E | F | G | H | I | J |   |  |
|  |  | M                                   | N | O | P | R | S | Š | T | U | V | Z | Ž | A | B | C | C | D | E | F | G | H | I | J | K |   |  |
|  |  | N                                   | O | P | R | S | Š | T | U | V | Z | Ž | A | B | C | C | D | E | F | G | H | I | J | K | L |   |  |
|  |  | O                                   | P | R | S | Š | T | U | V | Z | Ž | A | B | C | C | D | E | F | G | H | I | J | K | L | M |   |  |
|  |  | P                                   | R | S | Š | T | U | V | Z | Ž | A | B | C | C | D | E | F | G | H | I | J | K | L | M | N |   |  |
|  |  | R                                   | S | Š | T | U | V | Z | Ž | A | B | C | C | D | E | F | G | H | I | J | K | L | M | N | O |   |  |
|  |  | S                                   | Š | T | U | V | Z | Ž | A | B | C | C | D | E | F | G | H | I | J | K | L | M | N | O | P |   |  |
|  |  | T                                   | Š | U | V | Z | Ž | A | B | C | C | D | E | F | G | H | I | J | K | L | M | N | O | P | R |   |  |
|  |  | U                                   | V | Z | Ž | A | B | C | C | D | E | F | G | H | I | J | K | L | M | N | O | P | R | S | Š |   |  |
|  |  | V                                   | Z | Ž | A | B | C | C | D | E | F | G | H | I | J | K | L | M | N | O | P | R | S | Š | T |   |  |
|  |  | Z                                   | Ž | A | B | C | C | D | E | F | G | H | I | J | K | L | M | N | O | P | R | S | Š | T | U |   |  |
|  |  | Ž                                   | A | B | C | C | D | E | F | G | H | I | J | K | L | M | N | O | P | R | S | Š | T | U | V |   |  |

Tabela 7.3 Kriptiranje z Vigenèrjevim kriptogramom

celotna kodna tabela. Kot primer kodiranja lahko omenimo t.i. veliki kriptogram (angl. *great cipher*), ki se je v 17. stoletju uporabljal za skrivanje sporočil pred sovražnikom. Ta kriptogram na podoben način kodira posamezne zloge (franco-skega) jezika v številke, zanimiv prikaz njegovega zgodovinskega zapisa pa je prikazan na sliki 7.2.

### 7.2.2 Transpozicijske metode

Substitucijske metode zakrivajo vsebino sporočila s preslikovanjem znakov čistopisa v znake kriptograma, kljub temu pa ohranjajo urejenost znakov čistopisa in kriptograma: oba beremo od leve proti desni. Transpozicijske metode pristopajo h kriptiranju drugače, in sicer z zamenjavo vrstnega reda znakov v čistopisu, ne da bi posamezne znake zamenjevali z nadomestnimi.

Primer klasičnega transpozicijskega kriptograma je stolpični kriptogram, pri katerem kot ključ izberemo besedo ali frazo, ki ne vsebuje ponovljenih črk. Vsebino čistopisa nato prelomimo v več vrstic, od katerih vsaka vsebuje toliko črk,

| Nº                 | O   | P             | Q   | R                  | S    | T                     | V   | C   | I   | Z   | 8 |
|--------------------|-----|---------------|-----|--------------------|------|-----------------------|-----|-----|-----|-----|---|
| 811                | 117 | 219           | 447 | 511                | 355  | 346                   | 141 | 205 | 518 | 279 |   |
|                    | 238 |               |     |                    |      |                       | 169 |     | 422 | 648 |   |
| 702                | 559 | 538           | 595 | 723                | 627  | 613                   | 286 | 436 | 639 | 615 |   |
|                    | 500 |               |     |                    |      |                       | 364 |     |     | 827 |   |
| genera. t. uo.     | 155 | lais. x       | 668 | Ob.                | 19   | orange.               | 841 |     |     |     |   |
| genet.             | 555 | timins.       | 708 | aboi.              | 59   | patron. dre. tion. 30 |     |     |     |     |   |
| gen.               | 575 | lote.         | 728 | objets. id.        | 69   | precede.              |     |     |     |     |   |
| ges.               | 159 | le Roy de     | 758 | objets. ation.     | 79   | pre.                  |     |     |     |     |   |
| pla.               | 155 | le Prince de  | 798 | objets. ex. ation. | 179  | principal. uas        | 31  |     |     |     |   |
| gle.               | 215 | le Due de     | 928 | obstacle. s.       | 179  | privonneur.           | 152 |     |     |     |   |
| gli.               | 375 | le Marquis de | 938 | obtenir.           | 222  | pro.                  | 162 |     |     |     |   |
| glo. we            | 555 | le Baron de   | 988 | oc. auion.         | 249  | probain.              | 202 |     |     |     |   |
| gna.               | 375 | le Sieur de   | 149 | ocup. er           | 289  | profie. r.            | 262 |     |     |     |   |
| gne.               | 845 | loin.         | 79  | of.                | 349  | projec. r.            | 182 |     |     |     |   |
| gni.               | 485 | lon.          | 119 | office. leir. s.   | 429  | propagation.          | 202 |     |     |     |   |
| gno.               | 505 | lond.         | 189 | offre. s.          | 489  | provision. s.         | 112 |     |     |     |   |
| gouvern. g. men. e | 16  | tuy.          | 818 | oient.             | 499  | prow.                 | 162 |     |     |     |   |
| grage.             | 405 |               |     | air.               | 529  | pro.                  | 62  |     |     |     |   |
| grand.             | 525 |               |     | air.               | 559  | publier. c.           | 512 |     |     |     |   |
| gre.               | 585 |               |     | air.               | 629  | puiss. ances.         | 572 |     |     |     |   |
| gri.               | 625 |               |     | air.               | 669  |                       |     |     |     |     |   |
| gro.               | 665 |               |     | am.                | 729  | Off.                  | 641 |     |     |     |   |
| gua.               | 195 |               |     | an.                | 779  | qua.                  | 672 |     |     |     |   |
| gue.               | 235 |               |     | an.                | 819  | qualite.              | 311 |     |     |     |   |
| guerre.            | 225 |               |     | an.                | 849  | guard.                | 742 |     |     |     |   |
| gui. de. s.        | 895 |               |     | an.                | 849  | quantite.             | 741 |     |     |     |   |
|                    |     |               |     | an.                | 879  | guarante.             | 781 |     |     |     |   |
|                    |     |               |     | an.                | 899  | guarantir.            | 781 |     |     |     |   |
|                    |     |               |     | an.                | 929  | guaro. ier.           | 322 |     |     |     |   |
|                    |     |               |     | an.                | 959  | quatre.               | 582 |     |     |     |   |
|                    |     |               |     | an.                | 1009 | que.                  | 661 |     |     |     |   |
|                    |     |               |     | an.                | 1359 | ques. te. s.          | 182 |     |     |     |   |
|                    |     |               |     | an.                | 1659 | question. s.          | 23  |     |     |     |   |
|                    |     |               |     | an.                | 2109 | qui.                  | 55  |     |     |     |   |
|                    |     |               |     | an.                | 2409 | qui. d.               | 75  |     |     |     |   |
|                    |     |               |     | an.                | 2709 | quince.               | 193 |     |     |     |   |
|                    |     |               |     | an.                | 3009 | quo. n.               | 192 |     |     |     |   |
|                    |     |               |     | an.                | 3309 | 153                   |     |     |     |     |   |

**Slika 7.2** Kodiranje zlogov v številke [9]

kolikor jih ima ključ. Ključ nato uporabimo za to, da glede na abecedno urejenost črk v njem oštevilčimo zaporedje stolpcev. Čistopis dokončno pretvorimo v kriptogram tako, da izpišemo vsebino posameznih stolpcev. Primer uporabe transpozicijskega stolpičnega kriptograma je prikazan v tabeli 7.4.

Če bi s kriptoanalizo želeli razbiti sporočilo kriptograma, bi kriptoanalitik moral najprej ugotoviti, da ima opravka s transpozicijskim kriptogramom. To lahko preprosto ugotovi, saj so frekvence posameznih črk v velikih kriptogramih podobne frekvencam črk v celiem jeziku. V naslednjem koraku bi kriptoanalitik moral ugotoviti število stolpcev, pri čemer bi si lahko pomagal na ta način, da bi v sosednje stolpce "zlagal" takšne podnize kriptograma, ki tvorijo najbolj pogoste bigrame v jeziku. S poskušanjem različnih kombinacij bi lahko z uporabo jezikovnih značilnosti sčasoma odkril vsebino čistopisa.

Sodobne kriptografske metode uporabljajo kombinacijo pristopov preprostih metod, omenjenih v tem razdelku. Pri njih se je poudarek premaknil h kriptografskim algoritmom, za razumevanje katerih mora imeti tudi kriptoanalitik napredno razumevanje matematičnih osnov. V naslednjem razdelku se bomo posvetili dvema velikima družinama sodobnih kriptografskih algoritmov, **kriptografijs s simetričnimi ključi** in **kriptografijs z javnimi ključi**. Kot primera prvega pri-

| ključ                      | T                                                       | O | L | K | I | E | N |
|----------------------------|---------------------------------------------------------|---|---|---|---|---|---|
| vrstni red znakov v ključu | 7                                                       | 6 | 4 | 3 | 2 | 1 | 5 |
| čistopis                   | m                                                       | o | j | a | g | l | a |
|                            | v                                                       | a | j | e | k | o | t |
|                            | a                                                       | v | t | o | b | u | s |
|                            | v                                                       | e | d | n | o | j | e |
|                            | p                                                       | r | o | s | t | o | r |
|                            | z                                                       | a | š | e | e | n | o |
|                            | i                                                       | n | f | o | r | m | a |
|                            | c                                                       | i | j | o | a | b | c |
| <b>criptogram</b>          | loujonmbgkboteraaeonseoojjdošfjatseroacoaveranimvavpzic |   |   |   |   |   |   |

**Tabela 7.4** Kriptiranje s stolpičnim criptogramom

stopa si bomo pogledali algoritma DES (angl. *Data Encryption Standard*) in AES (angl. *Advanced Encryption Standard*), kot primer drugega pristopa pa metodo RSA (imenovana po avtorjih *Rivest-Shamir-Adleman*).

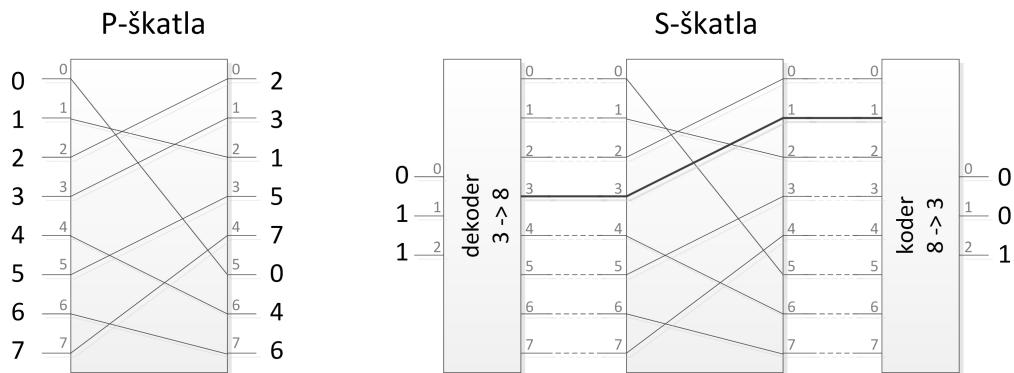
### 7.3 Data Encryption Standard (DES)

DES je kriptografska metoda, ki jo je ameriška vlada leta 1977 uveljavila kot uradni standard za kriptiranje. Metoda uporablja pristop **bločnega kriptiranja**, kar pomeni, da zaporedoma jemlje zaporedja (bloke) po  $n$  vhodnih bitov čistopisa, jih preslika v criptogram ter nato isti postopek ponovi z naslednjim zaporedjem n vhodnih bitov.

DES uporablja oba pristopa, ki smo ju spoznali v razdelku 7.2 – je kombinacija substitucijskih in transpozicijskih metod. Če želimo, lahko DES implementiramo programsko, s čimer dosežemo večjo fleksibilnost, ali pa strojno (z vezjem), s čimer dosežemo večjo hitrost izvajanja algoritma. Za naše potrebe bomo delovanje algoritma v nadaljevanju skicirali s konceptualnimi shemami.

Ker je transpozicija bitov definirana preprosto le kot permutacija bitov, pravimo, da ta postopek izvaja t.i. **permutacijska škatla** (ali *P-škatla*, angl. *P-box*). Primer P-škatle, ki preslika zaporedje bitov, ki ga označimo z 01234567 v 23157046 je prikazana na sliki 7.3 (levo). Izračunana permutacija (23157046), ki je lastna vsaki P-škatli, tvori torej njen enkripcijski ključ.

Drugi del algoritma, substitucijo, izvajajo **substitucijske škatle** (*S-škatle*, angl. *S-box*), ki pa so sestavljene iz treh delov: dekoderja, ki trojice bitov preslika v eno izhodno povezavo, permutacijskega vezja in koderja, ki izhodno povezavo preslika v izhodno trojico. Delovanje substitucijske škatle bomo najlažeje ponazorili na primeru, ki je skiciran na sliki 7.3 (desno). Vhodni dekoder 3/8 ima nalogu za



Slika 7.3 Primera P-škatle (levo) in S-škatle (desno)

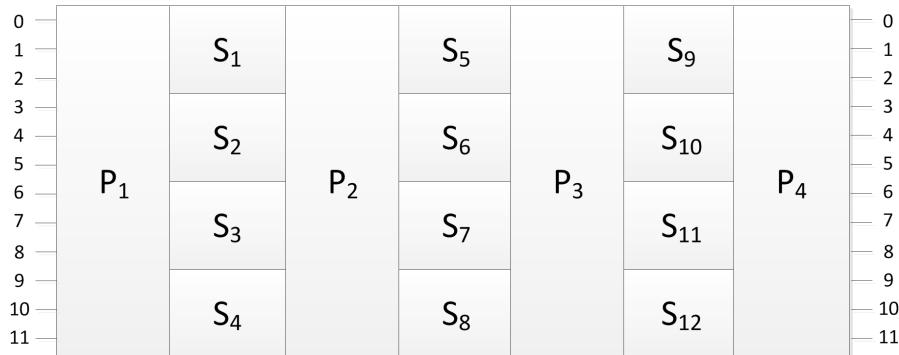
vhodno kombinacijo bitov 000 prižgati izhodni signal na ničtem izhodnem mestu (označimo krajše  $000 \Rightarrow 0$ ),  $001 \Rightarrow 1$ ,  $010 \Rightarrow 2$ , ... in  $111 \Rightarrow 7$ . Če imamo na vhodu torej niz 101, bo osrednje permutacijsko vezje signal 5 preslikalo v signal 3. Izhodni kodirnik 8/3 bo na ta način signal binarno kodiral v obratni smeri od dekoderja v 011. Opisani postopek nam implementira substitucijo, ki je prikazana v tabeli 7.5.

| čistopis | kriptogram |
|----------|------------|
| 000      | 101        |
| 001      | 010        |
| 010      | 000        |
| 011      | 001        |
| 100      | 110        |
| 101      | 011        |
| 110      | 111        |
| 111      | 100        |

Tabela 7.5 Substitucija, ki jo implementira S-škatla s slike 7.3

S kombiniranjem različnih P-škatel in S-škatel lahko implementiramo celo preslikovalno kaskado. Uporaba več manjših S-škatel namesto ene večje je priročna predvsem zato, ker potrebuje manj logike in spomina za implementacijo<sup>3</sup>. Kot primer si lahko ogledamo kaskado na sliki 7.4, ki je definirana z množico ključev za implementacijo P-škatel in S-škatel, ki kaskado sestavljajo.

<sup>3</sup>Če npr. primerjamo štiri S-škatle s tremi vhodi in eno z dvanajstimi (obe različici opravljata enako nalogo), lahko izračunamo, da manjše S-škatle s tremi vhodi hranijo  $4 \cdot 2^3 = 32$  preslikav, medtem ko jih ena večja hrani  $2^{12} = 4096$ .



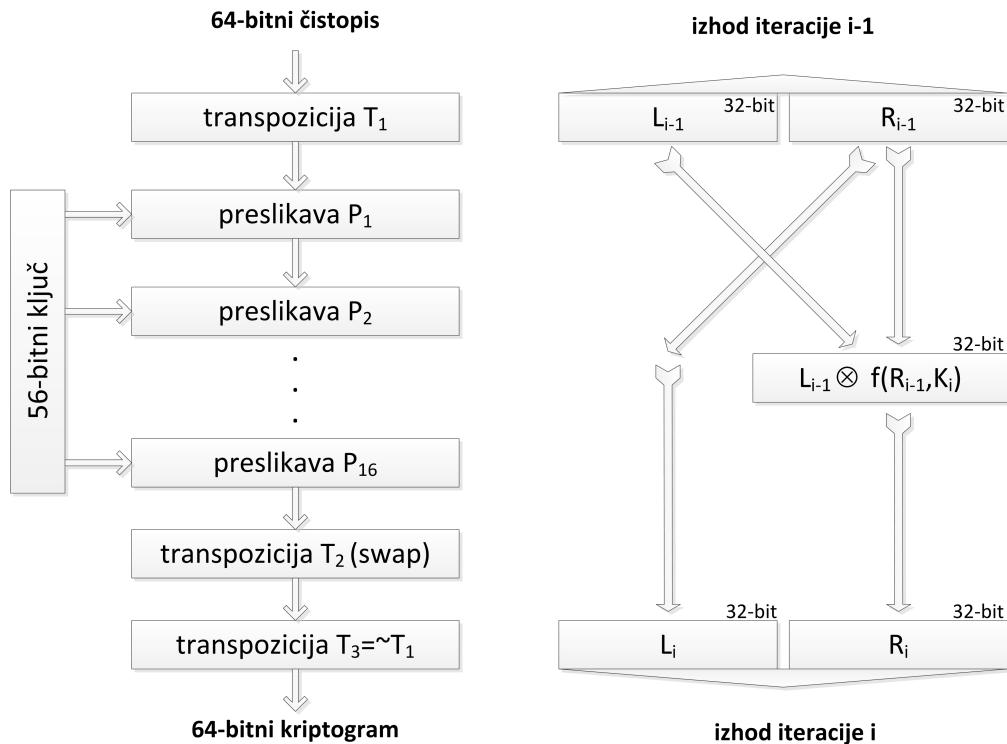
**Slika 7.4** Primer kaskade za kriptiranje 12-bitnih blokov, zložene iz štirih P-škatel in dvanajstih S-škatel

**Algoritem DES.** Z dosedanjim znanjem lahko končno opišemo delovanje algoritma DES. Ta razdeli vhodni čistopis na bloke po 64-bitov in zaporedno kriptira vsakega izmed njih, pri čemer uporablja 56-bitni ključ. Sam algoritem ima 19 faz, ki so v zaporedju:

- transpozicija  $T_1$ , neodvisna od enkripcijskega ključa,
- preslikave  $P_1, P_2, \dots, P_{16}$ , ki so parametrizirane z različnimi ključi, izpeljanimi iz glavnega 56-bitnega ključa,
- transpozicija  $T_2$ , ki zamenja levo in desno 32-bitno polovico izhoda (neodvisna od enkripcijskega ključa),
- transpozicija  $T_3$ , ki je inverzna transpoziciji  $T_1$  in tudi neodvisna od enkripcijskega ključa.

Celotna shema delovanja je prikazana na sliki 7.5 (levo), delovanje posamezne preslikave  $P_1, P_2, \dots, P_{16}$  pa na sliki 7.5 (desno). Iz desnega dela slike vidimo, da deluje vsaka preslikava tako, da desno polovico (32 bitov) preslika v levo polovico izhoda, novo desno polovico izhoda pa generira z izvedbo operacije XOR ( $\otimes$ ) med levo polovico vhoda in funkcijo, ki je parametrizirana z desno polovico vhoda in ključem.

Če vrednotimo kakovost algoritma DES, lahko poudarimo, da je možno permutacije in substitucije s strojno opremo implementirati zelo hitro, zato je algoritem uporaben pri večini aplikacij, ki delujejo v realnem času. Največ negativnih kritik v zvezi z algoritmom pa je glede njegove varnosti, saj se zaradi izrednega porasta računskih zmogljivosti DES ne obravnava več kot varen. S poskušanjem razbijanja kriptogramov so raziskovalne in druge organizacije v letu 1998 uspele razbiti kriptogram DES v 56 urah, v letu 1999 pa v dobrih 22 urah. Glede na porast strojnih kapacitet od zadnjega rezultata do danes lahko torej sklepamo, da potrebujemo v praksi varnejši algoritem od DES, algoritem



**Slika 7.5** Pregledna shema delovanja algoritma DES (levo) in shema delovanja posamezne preslikave  $P_i$  (desno)

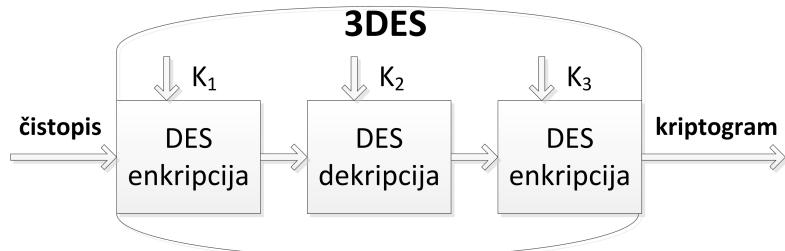
DES pa vendarle potrebujemo za združljivost s starejšimi sistemi, ki to različico še uporabljajo.

### 7.3.1 Trojni DES

V oktobru 1999 ameriški mednarodni standardizacijski biro nadomesti veljavo DES kot uradnega standarda za kriptiranje z njegovim naslednikom, algoritmom 3DES. 3DES ali *trojni DES* je dejansko algoritmom DES, uporabljen trikrat zapored s tremi različnimi ključi  $K_1$ ,  $K_2$  in  $K_3$ , kot to prikazuje slika 7.6.

Na sliki lahko vidimo, da se prvi in tretji zagon algoritma DES uporabita kot "enkripcija", kar pomeni, da algoritmom poženemo v originalnem vrstnem redu, kot smo ga predstavili v razdelku 7.3. Za razliko od tega pa drugi zagon izvedemo kot "dekripcijo", kar pomeni, da algoritmom poženemo v obratnem vrstnem redu. Razlog, zakaj je v algoritmu uporabljen zasnova kriptiranje-dekriptiranje-kriptiranje, tiči v združljivosti s sistemi, ki uporabljajo še stari (enojni) DES. Če uporabnika 3DES namreč omejimo samo na uporabo dveh ključev  $K_1$  in  $K_3$ , ter nastavimo, da je  $K_2 = K_1$ , potem predstavlja druga faza algoritma (dekripcija) inverz prve faze, rezultat česar je kriptiranje zgolj s ključem  $K_3$ . To

pomeni, da bosta oba algoritma na istih vhodih izračunala iste izhode oz. da je na ta način z algoritmom 3DES možno simulirati obnašanje algoritma DES.



**Slika 7.6** Kriptiranje z algoritmom 3DES

Ker sta v postopku uporabljeni še dve dodatni iteraciji kriptiranja s 56-bitnim ključem, naj bi bili kriptirani bloki  $2^{56}$ -krat varnejši. Istočasno pa je zaradi 3 iteracij izvajanja algoritmom 3DES trikrat počasnejši. Za obe različici algoritma DES je od nekdaj obstajal sum, da je vojska poznala matematično "bližnjico" za hitro razbijanje kriptogramov. Razvoj metod se je zato nadaljeval v smeri iskanja bolj varnih metod, kot je AES, ki ga omenjamo v nadaljevanju.

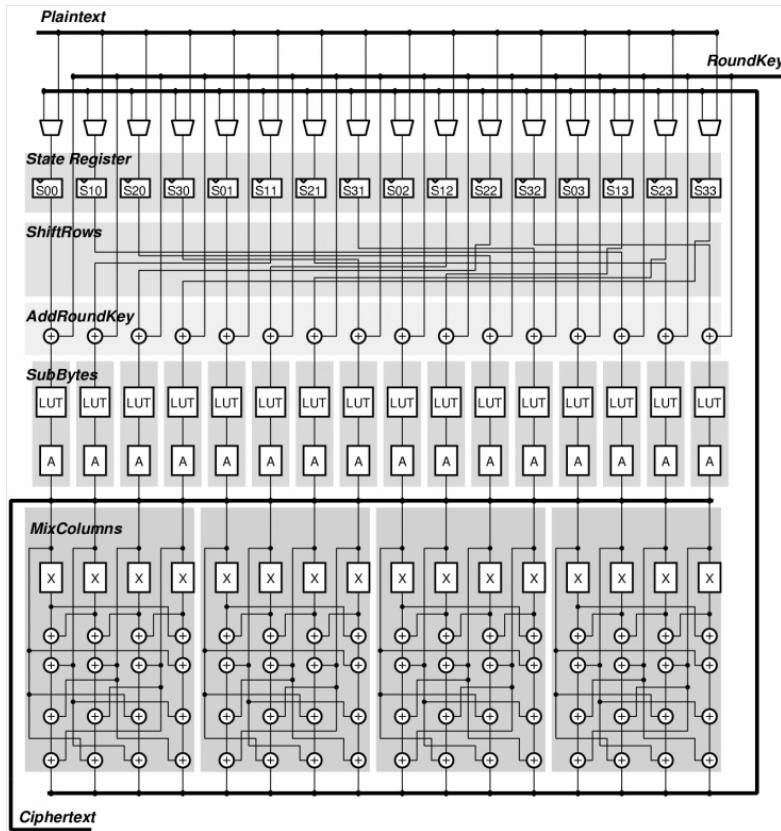
### 7.3.2 Advanced Encryption Standard (AES)

AES je nastal kot izboljšava algoritmov DES in 3DES, ko je ameriška Nacionalna agencija za standarde in tehnologijo (NIST) v letu 1997 izvedla javni razpis za zbiranje predlogov za novi kriptografski standard. Od petnajstih prispelih ponudb je oktobra 2000 je NIST izbral predlog, ki sta ga podala Joan Daemen in Vincent Rijment – algoritom, katerega ime je zloženka njunih priimkov, *Rijndael* (beri: [rajndo]).

Algoritom Rijndael podpira dolžine ključev in velikosti blokov od 128 do 256 bitov v korakih po 32 bitov. Čeprav je dolžina ključa in velikost kriptirnega bloka pri algoritmu Rijndael možno izbrati neodvisno, standard za AES določa, da mora velikost bloka biti natanko 128 bitov, velikost ključa pa 128, 192 ali 256 bitov. V praksi se AES najpogosteje uporablja s 128- ali 256-bitnim ključem in velikostjo kriptirnega bloka 128 bitov.

Podrobnosti delovanja algoritma AES presegajo obseg te knjige. Na kratko lahko omenimo, da ima algoritom 10-14 iteracij izvajanja in uporablja dodatne operacije, ki pri algoritmu DES niso prisotne: zamikanje vrstic, zamenjave stolpcev, izpeljave ključev. Iz globalne sheme delovanja tega algoritma na sliki 7.7 lahko razberemo, da je njegova kompleksnost delovanja občutno višja od kompleksnosti metode DES.

AES danes obravnavamo kot varno in učinkovito metodo. Ker je prostor ključev velik  $2^{128}$  elementov, se ocenjuje, da bi stroj z  $10^9$  vzporednimi procesorji



Slika 7.7 Konceptualna shema delovanja algoritma AES

in možnostjo preverjanja pravilnosti enega ključa v času  $10^{-12}$  s za razbijanje algoritma potreboval  $10^{10}$  let. Omenimo še, da je AES možno implementirati na hiter način, saj je zasnovan strojno in programsko učinkovito. Dobre implementacije dosegajo hitrost kriptiranja do 700Mbps, kar zadošča hitrosti za kriptiranje več kot 100 MPEG-2 video posnetkov v realnem času.

### 7.3.3 Verižno kriptiranje blokov

Verižno kriptiranje blokov (angl. *cypher block chaining*) ni še en nov kriptografski algoritem, pač pa metoda, ki odpravlja težavo vseh omenjenih bločnih metod (DES, 3DES in tudi AES). Pri bločnih metodah se namreč srečamo s pojavom, da se isti bloki čistopisa vedno preslikajo v iste bloke kriptogramov. Eden od načinov, kako lahko morebitni napadalec to izkoristi, je ta, da znane preslikave uporabi za lažje ugotavljanje vrednosti enkripcijskega ključa. Na aplikacijski plasti se bomo srečali s protokolom HTTP, pri katerem se sporočilo strežnikovega odgovora večino časa začne z nizom "HTTP/1.1 200 OK". Napadalec, ki bo opazil, da so začetki kriptiranih odgovorov HTTP vedno enaki zaporedju "INwb0L7YG84CP1Q",

bo s tem lahko omejil preiskovalni prostor enkripcijskih ključev in bo korak bližje k razbitju še preostalih delov sporočila.

Druga, še bolj verjetna možnost morebitne zlorabe znanih kriptogramov je ta, da lahko napadalec z njimi nadomesti druge dele kriptograma in s tem zamenja pravilno kriptirano vsebino z drugo pravilno kriptirano vsebino, česar prejemnik ne bo opazil. Kot primer opišimo izmišljen scenarij, kjer želi podjetje za sledenje maratonskim tekačem poslati rezultate zadnjega maratona organizatorju športne prireditve. Podjetje podatke posreduje tako, kot je prikazano v tabeli 7.6 (podatki so v resnici kriptirani, tabela prikazuje njihove čistopise): prvih 64+64 bitov<sup>4</sup> (2 bloka) podatkov zasedata ime in priimek tekmovalca, tretjih 64 bitov pa zaporedna številka mesta, ki ga je tekmovalec dosegel. V komunikacijo se vrine ljubosumni Janez, ki se na maraton ni pripravljal in je zato dosegel predzadnje mesto. Janez ve, da so podatki shranjeni v trojkah, zato lahko v kriptiranih rezultatih zamenja svoj tretji 64-bitni blok (zaporedno mesto na maratonu) za tretji blok Petra, ki je bil na tekmi drugi, in s tem ponaredi rezultate. Pri tem ni nujno, da Janez sploh pozna vsebino kriptograma, ampak lahko zamenjuje njihove bloke na slepo.

| ime                      |   |   |   |   |  |  |  | priimek                  |   |   |   |   |   |  |  | mesto                    |  |  |  |  |  |  |  |  |
|--------------------------|---|---|---|---|--|--|--|--------------------------|---|---|---|---|---|--|--|--------------------------|--|--|--|--|--|--|--|--|
| 64-bitni kriptirani blok |   |   |   |   |  |  |  | 64-bitni kriptirani blok |   |   |   |   |   |  |  | 64-bitni kriptirani blok |  |  |  |  |  |  |  |  |
| I                        | g | o | r |   |  |  |  | M                        | o | d | r | i |   |  |  | 1                        |  |  |  |  |  |  |  |  |
| P                        | e | t | a | r |  |  |  | V                        | e | l | i | k | i |  |  | 2                        |  |  |  |  |  |  |  |  |
| M                        | i | h | a |   |  |  |  | H                        | i | t | r | i |   |  |  | 3                        |  |  |  |  |  |  |  |  |
| J                        | a | n | a |   |  |  |  | D                        | r | z | n | a |   |  |  | 4                        |  |  |  |  |  |  |  |  |
| J                        | a | n | e | z |  |  |  | L                        | e | n | i |   |   |  |  | 5                        |  |  |  |  |  |  |  |  |

**Tabela 7.6** Primer, pri katerem lahko napadalec zamenja dva pravilno kriptirana bloka in s tem spremeni vsebino sporočila

Neprijetne opisane težave z bločnimi kriptografskimi metodami se znebimo tako, da uporabimo princip **verižnega kriptiranja blokov**, ki zagotavlja to, da bo isti niz vsakič kriptiran drugače. Mehanizem delovanja tega pristopa je prikazan na sliki 7.8. Slika prikazuje, kako pred kriptiranjem vsakega bloka čistopisa izvedemo operacijo XOR ( $\otimes$ ) čistopisa trenutnega bloka s kriptogramom prejšnjega bloka. Na ta način vpeljemo dodatno odvisnost trenutnih kriptogramov od zaporedja preteklih kriptogramov. Ker ob začetku postopka nimamo “prejšnjega bloka kriptograma”, s katerim bi lahko izvedli operacijo XOR s prvim blokom čistopisa, uporabimo za ta namen vrednost, ki jo imenujemo **inicializacijski vektor (IV)**. Gre za vrednost, ki jo poznata tako pošiljatelj kot prejemnik in jo lahko brez ogrožanja varnosti izmenjata v čistopisu pred začetkom kriptiranja.

<sup>4</sup>Predpostavljamo, da kodiramo črke po kodni tabeli, kjer je črka predpostavljena z 8 biti (npr. UTF-8).

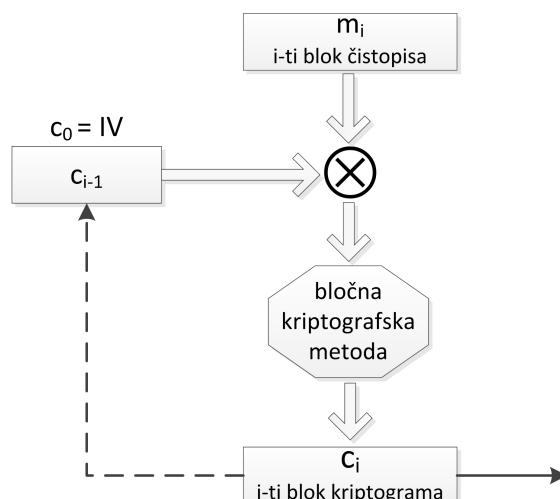
Če z  $m_i$  predstavimo  $i$ -ti zaporedni blok čistopisa, s  $c_i$   $i$ -ti izhodni blok kriptograma in znakom  $\otimes$  operacijo XOR, lahko metodo verižnega kriptiranja definiramo bolj formalno:

$$\begin{aligned} c_1 &= E(m_1 \otimes IV) \\ c_i &= E(m_i \otimes c_{i-1}) \end{aligned}$$

Prejemnik bo prejeto sporočilo odkriptiral z izvedbo operacij v obratnem vrstnem redu:

$$\begin{aligned} m_1 &= E(c_1) \otimes IV \\ m_i &= E(c_i) \otimes c_{i-1} \end{aligned}$$

Postopek bločnega kriptiranja je dovolj splošen, da ga lahko uporabimo s poljubnimi bločnimi kriptografskimi metodami, kot so DES, 3DES ali AES, ki smo jih že spoznali.



Slika 7.8 Postopek verižnega kriptiranja blokov

### 7.3.4 Varnost bločnih kriptosistemov

V zgodovini so se za razbijanje bločnih kriptografskih metod uporabljali različni pristopi. Omenimo lahko t.i. **diferenčno kriptoanalizo**, ki temelji na opazovanju sprememb v kriptogramih ob zelo majhnih spremembah v čistopisu. Na podlagi opazovanih razlik poskuša vdiralec sklepati na vsebino ključa in delovanje metode. Znani so tudi pristopi z **linearno kriptoanalizo**, kjer lahko napadalec z uporabo do  $2^{43}$  čistopisov omeji preiskovanje prostora ključev in razbije kriptogram. Zanimivi (eksotični!) pristopi temeljijo na opazovanju **porabe elektrike**,

saj je binarna enica (1) v logiki predstavljena z napetostjo 3V, ničla (0) pa z 0V. S poznavanjem števila enic v uporabljenem ključu lahko znatno omejimo možen prostor ključev. Podoben princip je tudi pristop z **merjenjem časa izvajanja** algoritma, saj različne operacije trajajo različno dolgo.

V splošnem lahko rečemo, da so bločne metode z daljšimi ključi bolj varne od tistih s krajsimi, saj daljši ključi omogočajo bolj kompleksno parametrizacijo algoritmov in s tem povečanje kompleksnosti kriptoanalize. Kot primer si oglejmo tabelo 7.7, kjer je prikazan ocenjen čas razbijanja bločnega sistema glede na uporabljeno dolžino ključa (stanje v letu 2013) s postopkom izčrpnega preiskovanja (s silo, angl. *brute force*).

| Ključ | Oseba      | Majhna skupina | Raziskovalno omrežje | Veliko podjetje | Vojska       |
|-------|------------|----------------|----------------------|-----------------|--------------|
| 40    | sekunde    | milisekunde    | milisekunde          | mikrosekunde    | nanosekunde  |
| 56    | dan        | ure            | ure                  | sekunde         | mikrosekunde |
| 64    | tedni      | tedni          | dnevi                | ure             | milisekunde  |
| 80    | tisočletja | tisočletja     | stoletja             | leta            | minute       |
| 128   | $\infty$   | $\infty$       | $\infty$             | $\infty$        | $\infty$     |

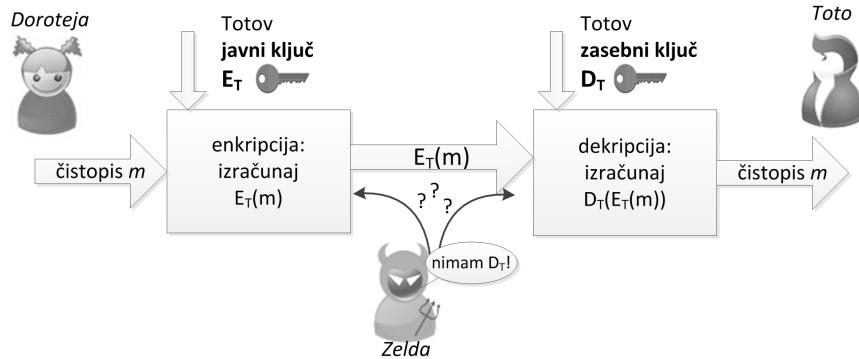
**Tabela 7.7** Ocena časa, potrebnega za razbijanje bločnega kriptosistema s silo glede na dolžino uporabljenega ključa (ocena avtorjev iz leta 2013)

## 7.4 Kriptografija z javnimi ključi

Največja težava pri bločnih kriptosistemih je **problem distribucije ključa**. Naj bo algoritom še toliko varen, težava se pojavi v točki, ko se morata pošiljatelj in prejemnik dogovoriti za skupni (simetrični) ključ, ki pa ga zaradi varnosti ne moreta poslati preko omrežja. Leta 1976 se je raziskovalcema Univerze v Stanfordu, Diffie-ju in Hellman-u, porodila drastično drugačna rešitev – razviti kriptosistem, ki bo za enkripcijo (pretvorbo čistopisa v kriptogram) uporabljal drugačen ključ  $E$  kot ključ  $D$ , ki bo namenjen dekripciji (pretvorbi kriptograma v čistopis). Ker omenjeni algoritom uporablja različna ključa za enkripcijo in dekripcijo, pravimo tovrstni kriptografiji **asimetrična kriptografija** ali **kriptografija z javnimi ključi**. Ta scenarij torej ustreza tistemu s slike 7.1, kjer velja  $K_1 \neq K_2$ .

Razlog, da tovrstno kriptografijo imenujemo tudi kriptografija z javnimi ključi tiči v načinu predvidene uporabe metode, ki jo pojasnimo na primeru, ki je prikazan na sliki 7.9. Če želi Toto prejemati podatke, ki so kriptirani in namenjeni samo njemu, lahko javno objavi enkripcijski algoritmom in njegov ključ  $E_T$  (Totove ključe označimo z indeksom  $T$ , torej  $D_T$  in  $E_T$ ). Pri tem svoj drugi ključ  $D_T$  ohrani tajnega samo zase. Doroteja, ki želi Totu poslati skrivno sporočilo  $m$ , ga bo torej zaskrila z izračunom  $E_T(m)$ . Toto je edini, ki pozna ključ  $D_T$  (edini ključ,

ki je inverzen ključu  $E_T$ ), zato bo lahko le on izračunal  $D_T(E_T(m)) = m$  in s tem razbral prvotni čistopis. Zaradi opisanega načina uporabe pravimo ključema  $E$  in  $D$  tudi javni ključ in zasebni (ali tajni) ključ.



**Slika 7.9** Uporaba javnega in zasebnega ključa pri kriptografiji z javnimi ključi

Zasnova zgornjega algoritma zveni vzpodbudno, težava pa se je pojavila pri tem, kako v praksi najti dovolj dober algoritem, ki bo deloval na opisan način. Leta 1978 so raziskovalci Rivest, Shamir in Adleman na univerzi M.I.T. predlagali algoritem RSA (poimenovan po začetnicah priimkov avtorjev), ki se je v zadnjih dobrih 30 letih izkazal kot varen. Zaradi velike dolžine ključa, ki ga algoritem RSA zahteva (1024 bitov), ima algoritem tudi slabost, saj je počasen. V nadaljevanju bomo opisali podrobnosti tega algoritma.

**Izbira parametrov algoritma RSA.** Delovanje algoritma lahko povzamemo v naslednjih korakih:

1. Izberemo dve veliki (običajno 1024 bitni) praštevili  $p$  in  $q$ .
2. Izračunamo števili  $n$  in  $z$  po pravilih
$$n = p \times q$$

$$z = (p - 1) \times (q - 1)$$
3. Izberemo število  $d$ , za katero velja, da je tuje (nima skupnih deliteljev) številu  $z$ .
4. Poiščemo takšno število  $e$ , za katerega velja  $(e \times d) \bmod z = 1$  (ostanek produkta  $e \times d$  pri deljenju z  $z$  je enak 1).

**Kriptiranje in dekriptiranje.** Ko smo določili števili  $d$  in  $e$ , razdelimo čistopis  $P$  na bloke, običajno daljše od 100 bitov. Ker črke čistopisa kodiramo z vrednostmi po neki kodni tabeli znakov, lahko sporočilo obravnavamo kot zaporedje teh vrednosti in za vsakega izračunamo kriptogram

$$C = P^e \bmod n$$

★ **Primer 7.3:** Preden nadaljujemo z opisom postopka kriptiranja in dekriptiranja, prikažimo primer zgornjega postopka. Za zgled izberimo majhni praštevili, s katerima bo lažje računati, denimo  $p = 11$ ,  $q = 17$ . Tedaj velja  $n = 187$  in  $z = 160$ . Za število  $d$  izberimo število 63, ki je tuje številu  $z$ . Poiskati moramo le še tak  $e$ , za katerega bo veljalo zgornje pravilo, torej  $e = \frac{k \cdot z + 1}{d} = \frac{k \cdot 160 + 1}{63}$  za nek  $k \in \mathbb{Z}$ . Poiskati moramo le še tak  $k$ , da bo rezultat ulomka celo število  $e$ . Z reševanjem enačbe ali poskušanjem lahko izberemo  $k = 50$ , ki da  $e = 127$ .

Kriptogram dekriptiramo po podobnem postopku:

$$P = C^d \pmod{n}$$

Vidimo, da sta za kriptiranje bila potrebna podatka  $(e, n)$ , za dekriptiranje pa podatka  $(d, n)$ . To sta torej podatka, ki pri metodi RSA v tem vrstnem redu predstavljata **javni ključ** in **zasebni ključ**.

Če želimo s parametri iz prejšnjega primera kriptirati črko q, ki je v kodnem naboru ASCII predstavljena z vrednostjo 113 (dvojiško: 01110001), jo kriptiramo v

$$C = 113^{127} \pmod{187} = 31$$

in dekriptiramo v

$$P = 31^{63} \pmod{187} = 113.$$

**Varnost in učinkovitost algoritma RSA.** Velja, da je algoritem RSA varen, ker v praksi ne obstaja učinkovit algoritem, ki bi lahko hitro poiskal delitelje zelo velikih števil. Če bi tak algoritem obstajal, bi morebitni napadalec lahko iz javno znanega  $n$  (ki je sestavni del javnega ključa) prišel do  $p$  in  $q$ , iz teh pa z nekaj matematike do  $e$  in  $d$ . Avtorji ocenjujejo, da bi iskanje deliteljev 500-mestnega števila z napadom s silo trajalo  $10^{25}$  let.

Ker je algoritem zaradi potenciranja zelo velikih praštevil počasen, se v praksi velikokrat uporablja le za kriptiranje krajsih sporočil, ki nimajo zahtev po kriptiranju v stvarnem (realnem) času. Pomemben problem, ki ga dobro rešujemo s kriptografijo RSA, navkljub njeni počasnosti, pa je distribucija ključev za veliko hitrejšo simetrično kriptografijo. Za konec poudarimo, da RSA nima težav z distribucijo ključev, saj je par števil  $(e, n)$ , ki tvori javni ključ, javno objavljen, enkripcijski algoritem pa zagotavlja, da je zasebni ključ  $(d, n)$  iz teh podatkov praktično nemogoče izračunati.

**Alternativni pristopi h kriptografiji z javnimi ključi.** Za konec razdelka omenimo, da obstajajo tudi drugi pristopi h kriptografiji z javnimi ključi. Takšen znan pristop je **kriptografija z eliptičnimi krivuljami** (angl. *elliptical curve cryptography*), ki je bila prvič omenjena leta 1985 in jo le na kratko opisujemo v

nadaljevanju. Eliptične krivulje (to niso elipse) so krivulje, definirane z množico točk  $y^2 = x^3 + ax + b$ . Te imajo lepo matematično lastnost, da premica, položena skozi poljubni dve točki (imenujmo ju  $A_1$  in  $A_2$ ), seka eliptično krivuljo tudi v neki tretji točki  $A_3$ . Če točko  $A_3$  vzamemo za naslednjo točko v zaporedju, skozi katero položimo premico, ki istočasno potuje tudi skozi začetno točko  $A_1$ , dobimo novo presečišče  $A_4$ . Ta postopek, ki spominja na odbijanje biljard krogle ob robove mize, lahko ponovimo  $n$  krat in s tem pridemo v končno točko  $A_n$ . Velja, da samo iz začetne točke  $A_1$  in končne točke  $A_n$  težko ugotovimo, koliko je  $n$ , zato lahko  $n$  izberemo kot primeren zasebni ključ,  $A_n$  pa kot javni. Ta princip je soroden principu RSA, ki temelji na tem, da je množenje (potenciranje) praštevil preprosto, razstavljanje na delitelje pa računsko težko. Problem iskanja števila  $n$  je matematično težji v primerjavi z iskanjem deliteljev velikih števil in zato naj bi bila tudi eliptična kriptografija bolj varna. Raziskave kažejo, da je kriptografija z eliptičnimi krivuljami s 164-bitnim ključem enako varna kot ostale sodobne metode, ki uporabljajo 1024-biten ključ. Kljub večji varnosti in posledično večji hitrosti (krajši ključ), ostaja metoda RSA bolj popularna zaradi razumljivosti postopka in lažje implementacije; obstaja tudi sum, da je določen nabor eliptičnih krivulj manj varen od ostalih, zato ima industrija pri uporabi te metode še zadržke.

## 7.5 Pomožni kriptografski postopki

### 7.5.1 Zgoščevalne funkcije

Za uspešno zagotavljanje varnosti potrebujemo poleg kriptografskih algoritmov še nekaj matematičnih postopkov, ki jih bomo opisali v nadaljevanju. V preostanku tega poglavja bomo kasneje opisali tudi način in namen njihove uporabe, na primer za generiranje ključev ali za izvajanje avtentikacijskih postopkov. Namen tega razdelka pa je predvsem opis načina delovanja.

Zgoščevalne funkcije (imenujemo jih tudi *hash*, razprševalne funkcije, digitalni prstni odtis, *fingerprint*, *footprint*) so enosmerne funkcije, ki poljuben vhodni čistopis preslikajo v t.i. zgoščeno vrednost. Za idealne zgoščevalne funkcije bi veljalo, da so injektivne, t.j. da nobena dva vhodna čistopisa ne preslikajo v isto zgoščeno vrednost. Vendar pa v računalniških komunikacijah pogosto potrebujemo zgoščene vrednosti, ki so točno določene dolžine, saj je za zgoščeno vrednost v aplikacijskih sporočilih lahko predvideno točno določeno število bitov. Zato v praksi uporabljamo takšne zgoščevalne funkcije, ki lahko na vhodu sprejemajo neskončno število različnih vrednosti (čistopisov), te pa preslikajo v množico končne velikosti. Ker se več različnih vhodnih čistopisov lahko preslika v isto zgoščeno vrednost, takšne funkcije torej niso injektivne.

Primer zelo preproste 4-mestne desetiške zgoščevalne funkcije, ki deluje nad nizom števk, je vsota števk. Seštejemo vse števke iz vhodnega niza in če je njihova vsota več kot 4-mestna (torej večja od 9999), postopek ponovimo

nad vsoto. Prstni odtis niza 4257125840002 je tako 40. Prav tako je prstni odtis niza 88888 enak 40 in enak je tudi prstni odtis niza 4444444444. Nizov s prstnim odtisom 40 bi lahko našteli še veliko, pravzaprav jih je neskončno mnogo. Zgoščevalne funkcije zato ne moremo uporabiti v inverzu, torej za računanje osnovnega niza iz zgoščene vrednosti. Možnih rezultatov bi bilo namreč neskončno.

Dobre zgoščevalne funkcije vrednosti enakomerno razpršijo po prostoru. To pomeni, da sta zgoščeni vrednosti dveh zelo podobnih vhodnih nizov praviloma zelo različni in da je pri vseh možnih različnih vhodnih vrednosti frekvenca vseh rezultatov enaka.

Za dano zgoščeno vrednost mora biti tudi zelo težko najti ustrezzo vhodno vrednost. Ko se dve različni vhodni vrednosti preslikata v isto izhodno vrednost, rečemo temu kolizija. Iskanje kolizije predstavlja najresnejšo obliko napada na zgoščevalne funkcije, kot bomo videli v nadaljevanju. Pri našem primeru vsote števk predstavljajo kolizijo vsi omenjeni nizi števk, ki imajo prstni odtis enak 40.

Po svojem načinu delovanja so zgoščevalne funkcije močno podobne sodobnim simetričnim kriptografskim metodam, saj vhodni niz razdelijo na bloke določene dolžine, nad katerimi nato izvajajo razmeroma preproste bitne operacije. Imajo tudi podobne slabosti. Zgoščevalne funkcije z rezultatom manjše dolžine, na primer 56 bitov, so bolj ranljive za napad z grobo silo kot tiste z rezultatom večje dolžine, na primer 128 ali celo 256 bitov. Z besedo napad tu mislimo iskanje kolizije, torej iskanje vhodne vrednosti, ki se preslika v opazovani 56-bitni (ali 160-bitni ali 256-bitni) zgoščeni niz. Za napad z grobo silo na 56-bitni zgoščeni niz bi v povprečju potrebovali  $2^{56}/2$  poskusov, za napad na 160-bitni zgoščeni niz pa  $2^{160}/2$  poskusov.

Danes sta najpogosteje uporabljeni zgoščevalni funkciji MD5 in SHA-1. MD5 враča kot rezultat 128-bitno zgoščeno vrednost, SHA-1 pa 160-bitno. Njun princip delovanja je podoben. Vhodni binarni niz obe razdelita na zaporedje 512-bitnih blokov. Če se število bitov ne izide in je zadnji blok krajši, se mu doda določene bite do dolžine 512. Temu se nato prišteje število, ki ponazarja dolžino celotnega vhodnega niza. Nato se procesira vsak blok posebej z bitnimi operacijami, podobnimi kot v simetrični kriptografiji. Rezultate se med seboj sešteva po modulu in reducira, tako da dobimo na koncu, ko zmanjka 512-bitnih blokov, izhodni niz ustrezne dolžine.

Algoritma MD5 danes ne štejemo več za varnega, ker so raziskovalci pokazali, da so možni napadi, ki znižajo zahtevnost napada z grobo silo za nekaj velikostnih razredov [3]. Zato ga je priporočljivo uporabljati le v implementacijah z nižjo varnostno občutljivostjo. SHA-1 imamo danes za varnega, vendar pa se za res občutljive namene uporabe priporoča novejši SHA-256, ki враča 256-bitno zgoščeno vrednost, predlagana pa sta že tudi SHA-384 in SHA-512.

Zgoščeno vrednost navadno pripnemo k sporočilu zato, da bi dokazali nje-govo avtentičnost, integriteto ali pa onemogočili zanikanje. Ker pa to lahko

naredi tudi napadalec, se je že kmalu pokazala potreba po zgoščevalni funkciji, ki bi delovala (podobno kot kriptografske metode) s ključem. Uporaba pravega tajnega ključa bi tako sogovornikoma v komunikaciji zagotavljala tudi avtentičnost zgoščene vrednosti. Kot bomo videli v nadaljevanju, lahko za ta namen uporabimo kar tajne asimetrične ključe iz sistema PKI (angl. *Public Key Infrastructure*, kriptografska infrastruktura javnega ključa), če pa tega ne uporabljamo, imamo na voljo postopek MAC (angl. *Message Authentication Code*). Najpogosteje uporabljeni MAC se imenuje HMAC (angl. *keyed-Hashing for Message Authentication*), njegove lastnosti pa so:

- uporaba obstoječih, dobro znanih in preizkušenih zgoščevalnih funkcij,
- možnost menjave ene zgoščevalne funkcije z drugo, če se npr. izkaže potreba po varnejši,
- ohranjanje učinkovitosti (hitrosti) računanja, kot je pri uporabljeni zgoščevalni funkciji,
- možnost preprostega upravljanja s ključi,
- dobro razumevanje, kako močna je avtentikacija (s kriptografskega vidika).

Načina delovanja HMAC na tem mestu ne bomo podrobno opisovali, gre pa za nekaj relativno preprostih bitnih operacij nad dvema parametromi – ključem in zgoščeno vrednostjo sporočila, ki jo vrne vgrajena zgoščevalna funkcija.

### 7.5.2 Generatorji naključnih števil in generatorji praštevil

Podobno kot zgoščevalne funkcije sodi med nujno potrebne matematične postopke za zagotavljanje varnosti tudi generiranje naključnih števil. V kriptografskih postopkih igrajo naključna števila pomembno vlogo. Uporabljamo jih pri generiranju asimetričnih ključev RSA, simetričnih sejnih ključev, pri generiranju izzivov za avtentikacijo in podobno. Generiranje ključnih števil je temelj varnostnih mehanizmov in s slabim generatorjem naključnih števil nam ne more pomagati niti še tako močan kriptografski postopek, saj bo napadalec lahko uganil ključ.

Če bi napadalcu uspelo uganiti naše naslednje "naključno" število, ki je predstavljalo temelj za izračun ključa, bi se lahko hitro dokopal tudi do samega ključa. Zato moramo poskrbeti, da naključna števila res dajejo vtis naključnosti in da jih ni možno vnaprej predvidevati.

Za zaporedje števil lahko rečemo, da je naključno, če velja dvoje:

1. Števila so **enakomerno porazdeljena** oziroma frekvenca pojavljanja števil je približno enaka. To lahko preverimo relativno preprosto.
2. **Neodvisnost**: nobenega števila ne moremo uganiti ali sklepati na njegovo vrednost na podlagi predhodnih števil v zaporedju.

Statistika pozna dobro definirane metode, s katerimi lahko dokažemo, da neko zaporedje ustreza določeni porazdelitvi. Žal pa ne pozna testa, ki bi dokazal,

da ne ustreza nobeni porazdelitvi (tj. da so števila med seboj neodvisna), torej ni testa neodvisnosti. Zato neodvisnost navadno preverjamo tako, da uporabimo dovolj veliko število različnih testov porazdelitve, da smo potem, ko vsi vračajo negativne rezultate, dovolj prepričani, da ni odvisnosti med števili.

Naključna števila, ki jih vrača generator naključnih števil, morajo biti **nepredvidljiva**: na podlagi prejšnjih števil ne sme biti možno napovedati naslednjega. Viri resnično naključnih števil so redki in temeljijo na naključnih dogodkih: zajemamo jih lahko na primer v fizičnih generatorjih šuma, v detektorjih delcev pri ionizirajočih sevanjih, pri puščajočih kondenzatorjih in podobno. Generatorji naključnih števil generirajo psevdonaključna števila, torej taka, ki so videti naključna, vendar pa jih je generiral nek algoritem. Zato je spet potrebno preverjati, ali se na podlagi prejšnjih števil lahko kakorkoli sklepa na naslednje število. Takšna zaporedja števil so deterministična in števila statistično gledano niso naključna, saj se po določeni periodi zaporedje števil začne ponavljati. Pravilo, s pomočjo katerega generiramo psevdonaključna števila, opišemo z iterativno formulo, ki ima splošno obliko, kot sledi:

$$X_{n+1} = (aX_n + C) \mod m.$$

$X_0$  je začetni člen zaporedja in imenujemo ga seme (angl. *seed*). Če so števila  $a$ ,  $C$  in  $m$  cela števila, potem formula vrača zaporedje celih števil med 0 in  $m$ . Za dober psevdonaključni generator je ključno, da so ta števila izbrana pametno, da je perioda velika in da je tudi sam  $m$  velik – blizu največjega nenegativnega celega števila, ki ga lahko predstavimo na konkretnem računalniku. Ta vrednost je tako lahko reda velikosti okoli  $2^{31}$  ali  $2^{63}$ .

## 7.6 Avtentikacija pošiljatelja in prejemnika

V vsakdanjem življenju potrebujemo pri komuniciranju dokaz, da res komuniciramo z želenim prejemnikom. Ravno tako potrebuje tudi prejemnik dokaz, da je pošiljatelj res oseba, za katero jo ima. Ker imamo pri računalniški komunikaciji težavo, da našega sogovornika ne moremo videti, pomeni, da moramo izvesti njegovo **avtentikacijo** (preverjanje in potrjevanje njegove identitete) na drugačen način. Tovrstne situacije srečamo npr. pri spletnem nakupovanju, kadar potrebujemo zagotovilo, da je na drugi strani res verodostojen spletni prodajalec in ne ponarejena spletna stran, namenjena zbiranju številk kreditnih kartic.

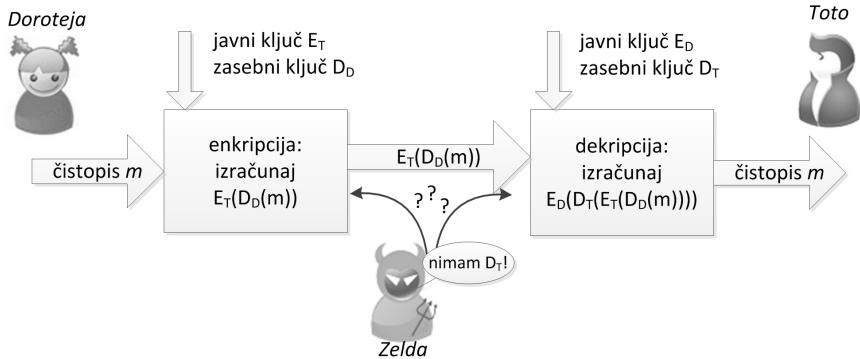
V razdelku 7.4 smo spoznali kriptografijo RSA, ki uporablja javni ključ  $E$  in zasebni ključ  $D$ . Ker je zasebni (dekripcijski) ključ  $D$  znan le njegovemu lastniku, se nam poznavanje tega ključa ponuja kot primerna metoda za preverjanje identitete prejemnika. Poglejmo si tri načine implementacije takšne avtentikacije:

- **avtentikacija pošiljatelja:** kadar želi pošiljatelj  $A$  dokazati, da je sporočilo  $m$  res poslal on, ga lahko kriptira s svojim zasebnim ključem in prejemniku

B pošlje kriptogram  $D_A(m)$ . Ker je javni ključ pošiljatelja javno dostopen (npr. na uporabnikovi spletni strani), lahko prejemnik sporočila z javnim ključem dekriptira in prebere sporočilo, torej izračuna  $E_A(D_A(m)) = m$ . Če prejemnik iz sporočila ponovno dekriptira berljivo vsebino, ima dokaz, da je sporočilo res poslal uporabnik A. Ker je zasebni ključ  $D_A$  znan edino uporabniku A, je lahko le on pripravil sporočilo  $D_A(m)$ , ki je berljivo z njegovim javnim ključem. Čeprav smo s to metodo zagotovili avtentikacijo pošiljatelja, pa je potrebno opozoriti, da lahko morebitni prisluškovalec prestreže sporočilo in ga dekriptira enako, kot to lahko stori namenski prejemnik B. Za zakrivanje vsebine pred nepovabljenimi očmi potrebujemo torej še dodatne mehanizme;

- **avtentikacija prejemnika:** na podoben način, kot postopek avtentikacije pošiljatelja izkorišča tajnost njegovega zasebnega ključa, izkorišča postopek avtentikacije prejemnika enako lastnost. Če želi pošiljatelj poskrbeti, da lahko njegovo sporočilo gledajo le pooblaščene oči, kriptira svoje sporočilo  $m$  z javnim ključem prejemnika B, tako da dobi  $E_B(m)$ . Ker lahko le prejemnik izračuna  $D_B(E_B(m)) = m$ , lahko sporočilo bere samo on. Opisani postopek je prikazan na sliki 7.9 in varuje pred nepovabljenimi očmi, vendar pa ni odporen na ponarejanje sporočil, saj ne preprečuje morebitnemu napadalcu, da se predstavi z lažno identiteto in (na sicer varen način) komunicira s prejemnikom, kot da je nekdo drug;
- **vzajemna avtentikacija pošiljatelja in prejemnika:** ker smo videli, da javne ključe lahko uporablja kdorkoli (tudi morebitni napadalec), zgoraj opisana komunikacija ni v celoti varna. Če želimo poskrbeti za obojestransko avtentikacijo, uporabimo kriptografijo z javnimi ključi in sporočilo  $m$  kriptiramo dvojno, kot je prikazano na sliki 7.10. Za kriptiranje uporabimo ključa, ki sta znana pošiljatelju A in izračunamo  $E_B(D_A(m)) = D_A(E_B(m))$ . Prejemnik na prejetem dvojnem kriptogramu uporabi svoj zasebni ključ (s čimer avtenticiramo prejemnika) in javni ključ pošiljatelja (s čimer avtenticiramo pošiljatelja). Prejemnik torej dvojni kriptogram dekriptira z izračunom  $E_A(D_B(E_B(D_A(m)))) = m$ . V primerih, ko je sporočilo m dolgo, je opisan postopek zamuden in nepraktičen in raje uporabljamo postopek elektronskega podpisa, ki ga bomo spoznali v razdelku 7.7.3.

Omenimo, da kriptiranje celih sporočil z metodo RSA ni edini pristop, ki omogoča izvajanje avtentikacije med pošiljateljem in prejemnikom. Poznamo več protokolov, ki se uporablajo za ta namen, in jih bomo podrobneje spoznali v naslednjih razdelkih.



**Slika 7.10** Postopek obojestranske avtentikacije z metodo RSA

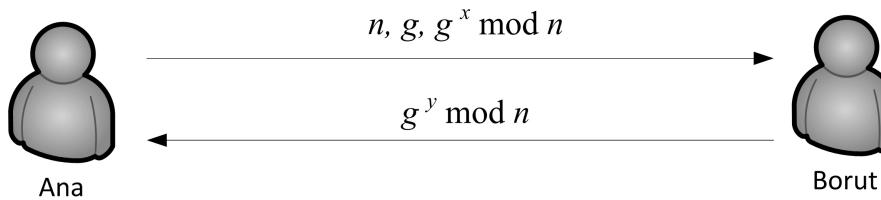
### 7.6.1 Vzpostavljanje skupnega ključa: protokol Diffie-Hellman

Do sedaj smo že omenili, da se strani, ki komunicirata med seboj (npr. pri uporabi simetrične kriptografije), vnaprej dogovorita za simetrični enkripcijski ključ. Pri preučevanju avtentikacijskih protokolov bomo tudi v nadaljevanju večkrat predpostavili, da je vzpostavitev skupnega ključa, ali kot tudi rečemo bolj splošno, vzpostavitev skupne skrivnosti, možna tudi preko ne-varnega omrežja. V tem razdelku bomo predstavili enega od protokolov za vzpostavitev takšnega ključa, ki je po svojih avtorjih imenovan kot protokol Diffie – Hellman.

Ana si zamisli število  $n$  in  $g$ , ki ju ni potrebno skrivati pred morebitnim napadalcem. Upoštevati mora naslednja pogoja:

1.  $n$  je praštevilo,
2. za  $g$  velja, da so vsa števila v zaporedju  $g \pmod n, g^2 \pmod n, g^3 \pmod n, \dots, g^{n-1} \pmod n$  različna in da zavzamejo vse vrednosti iz zaporedja  $1, 2, \dots, n - 1$  (v kakršnem koli vrstnem redu). Za tak  $g$  pravimo, da je primitivni koren števila  $n$  in poznamo matematične postopke, ki jih uporabljamo za iskanje takšnih števil.

Potem si Ana izbere število  $x$ , ki ga skrbno čuva pred nepovabljenim očesom. Izračuna vrednost  $g^x$  in nato še njen ostanek po deljenju z  $n$ . Rezultat skupaj s števili  $n$  in  $g$  pošlje v čistopisu Borutu. Borut si izbere število  $y$ , ki ga ravno tako dobro skriva. Izračuna vrednost  $g^y$  in nato še njen ostanek po deljenju z  $n$ , rezultat pa vrne Ani. Vsak sam pri sebi nato lahko izračuna ključ oziroma skupno skrivnost tako, da prejeti rezultat zmnoži z  $g^x$  (Ana) oziroma  $g^y$  (Borut) in še enkrat izračuna ostanek po deljenju z  $n$ . Rezultat računanja je na obeh straneh enak, morebitni prisluškovalec pa bi moral na podlagi prestreženih vrednosti rešiti diskretni logaritem, kar je težak računski problem in ga imamo pri dovolj velikih številih za neizračunljivega z današnjo računsko močjo. Postopek je prikazan na sliki 7.11.

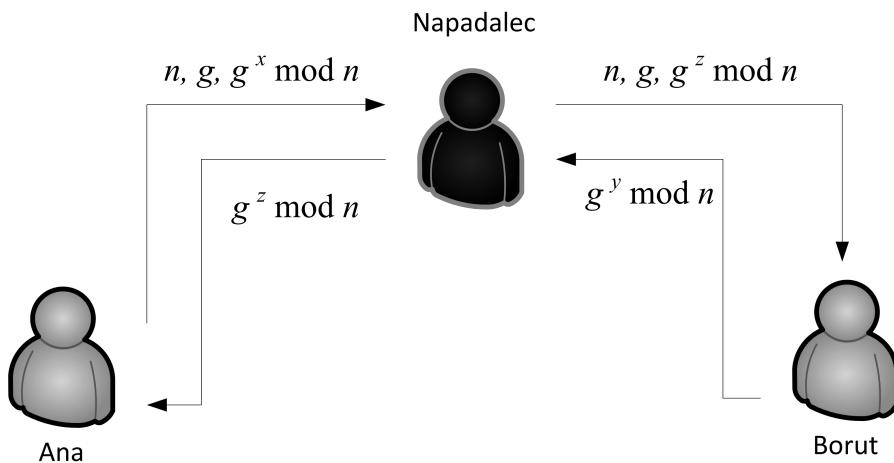


$$K = g^{xy} \text{ mod } n$$

$$K = g^{xy} \text{ mod } n$$

**Slika 7.11** Protokol Diffie – Hellman za varno vzpostavitev skupne skrivnosti ali simetričnega ključa

Žal pa opisani protokol ni popolnoma varen. Če napadalec prisluškuje komunikaciji ravno v pravem trenutku, da prestreže in zadrži Anino prvo sporočilo, lahko izvede napad s prestrezanjem (*man-in-the-middle*), kot prikazuje slika 7.12. Borutu napadalec namesto Anine izračunane vrednosti  $g^x \text{ mod } n$  pošlje svojo različico  $g^z \text{ mod } n$ . Ko Borut odgovori po protokolu, misleč da komunicira z Ano, pravzaprav pa vzpostavi ključ  $K = g^{xz} \text{ mod } n$  z napadalcem. Ta posreduje svojo različico  $g^z \text{ mod } n$  še Ani, kot da ji jo pošilja Borut, in tako vzpostavi drugi ključ  $K = g^{yz} \text{ mod } n$  še z Ano. Vsa nadaljnja komunikacija bo potekala prek napadalca, Borut in Ana pa neposredno sploh ne bosta mogla komunicirati, saj nimata vzpostavljenega skupnega ključa.



$$K = g^{xz} \text{ mod } n$$

$$K = g^{yz} \text{ mod } n$$

**Slika 7.12** Ranljivost protokola Diffie-Hellman: napadalec se lahko vrine v komunikacijo in vzpostavi skupna ključa z Ano in Borutom, ne da bi onadva vedela

### 7.6.2 Avtentikacijski protokoli

Avtentikacija je postopek preverjanja identitete sogovornika v komunikaciji. S pomočjo avtentikacijskih protokolov se tako lahko vsak od sogovornikov prepriča, s kom pravzaprav komunicira. Izziv dobrega načrtovanja avtentikacijskih protokolov je v tem, da tečejo prek ne-varnega omrežja – divjega interneta. Vsako sporočilo lahko tako sliši, prestreže in spremeni morebitni napadalec.

V tem razdelku bomo pregledali tri skupine avtentikacijskih protokolov in pokazali, kateri element jim daje varnost oziroma kako trdno je lahko uporabnik prepričan, da res komunicira s pravim sogovornikom.

Če za hip pozabimo elektronske komunikacije in si predstavljamo sobo, polno ljudi, poteka avtentikacija tako, da vsak preprosto pove sogovorniku svoje ime. Ker ponavadi ljudje ne lažejo, informaciji kar verjamemo. Napad na tak protokol je preprost: napadalec se predstavi s poljubnim lažnim imenom, sogovornik pa nima načina, da njegovo ime preveri. Če želimo take napade preprečiti, lahko uporabimo uradne policijske metode: ob predstavljanju preverimo osebno izkaznico sogovornika in se tako prepričamo, da tudi na njej piše isto ime. Napad na ta izboljšani protokol je že precej težji, saj si mora napadalec vnaprej priskrbeti osebno izkaznico z imenom, ki bi ga rad prevzel za svojo lažno identiteto.

V nadalnjem besedilu bomo privzeli, da sta sogovornika računalniška procesa, ki komunicirata prek Interneta, na primer spletni strežnik in odjemalec – brskalnik. Odjemalec se strežniku lahko predstavi z uporabniškim imenom, kot dokaz svoje identitete pa pošlje še skrivno geslo. Težava pri geslu (če se pošilja v nekriptirani obliki) je seveda ta, da lahko napadalec, ki prисluškuje komunikaciji, shrani prestreženo geslo in ga naslednjič uporabi, da se sam avtenticira z lažno identitetom nič hudega slutečega odjemalca. Da bi to preprečil, se odjemalec lahko odloči, da bo geslo pošiljal zakrito, torej v kriptirani obliki. Vendar celo tako geslo lahko napadalec prestreže in uporabi v naslednji komunikaciji. Res je, ne zna ga dekriptirati in ponovno kriptirati, vendar ga lahko v enaki obliki, kot ga je prestregel, predvaja strežniku – torej kriptiranega, kot dokaz svoje lažne identitete. Napad spada v skupino napadov s posneto sejo (angl. *playback attack* ali *replay attack*).

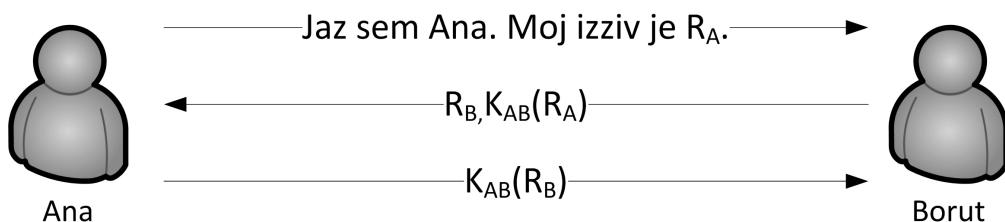
Tovrstne napade lahko preprečimo le, če za dokazovanje identitete vsakokrat uporabimo drugačno informacijo, na primer enkratna gesla. Na podobni osnovi delujejo tudi protokoli tipa *izziv – odgovor*, ki jih bomo predstavili v nadaljevanju.

Alternativno pa lahko za preverjanje identitete zadolžimo neko osrednjo avtoritetu, ki ji zaupamo vsi potencialni udeleženci komunikacije. Tudi protokole, ki uporabljajo osrednjo avtoritetu, bomo pojasnili v naslednjih razdelkih.

### Protokoli tipa izliv – odgovor

Avtentikacijski protokoli tipa izliv-odgovor predpostavljajo, da sta se sogovornika na varen način že vnaprej dogovorila za ključ, ki ga bosta uporabila kot dokaz avtentičnosti svoje identitete. Uporablja se hitra simetrična kriptografija, zato zadošča en ključ za avtentikacijo obeh sogovornikov. Ta ključ predstavlja njuno skupno skrivnost, zato tak način avtentikacije imenujemo tudi avtentikacija s skupno skrivnostjo (angl. *shared secret*). Ker sta sogovornika vzpostavila ključ na varen način, sta prepričana, da ga nima nihče drug razen njiju. Torej: kdor me prepriča, da ima ta skupni ključ, je moj pravi sogovornik in je s tem dokazal svojo identiteto.

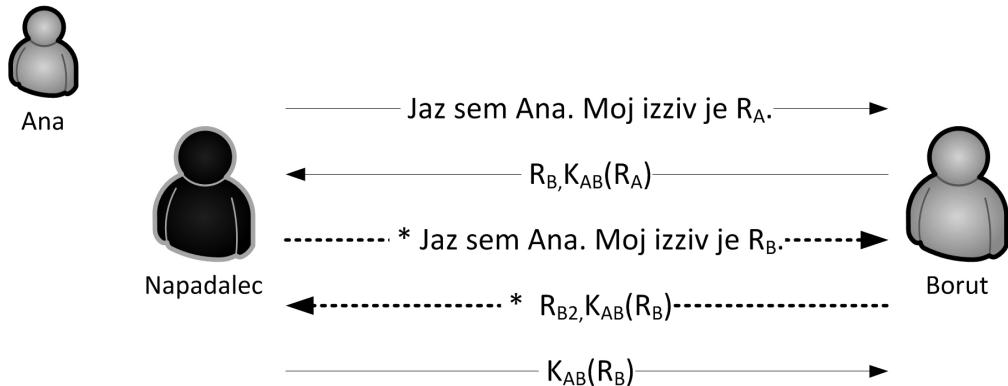
Najpreprostejša oblika protokola izliv – odgovor je tako naslednja: Ena stran pošlje drugi naključno generirano število, ki predstavlja izliv. Druga stran mora ta izliv kriptirati s skupnim ključem in kriptiranega poslati nazaj. Prva stran preveri, ali je prejela pravilno kriptiran izliv. Če je odgovor pravilen, potem je druga stran dokazala svojo identiteto. Primer takšnega protokola, kjer se avtenticirata oba sogovornika, prikazuje slika 7.13.  $R_A$  je izliv, ki ga pošlje Ana,  $R_B$  pa izliv, ki ga pošlje Borut. Pravilen odgovor na izliv  $R_A$  je s skupnim ključem  $K_{AB}$  kriptiran izliv  $R_A$  – s tem Borut dokaže svojo identiteto Ani. In obratno: Ana dokaže svojo identiteto Borutu tako, da mu vrne s skupnim ključem  $K_{AB}$  kriptiran izliv  $R_B$ .



**Slika 7.13** Primer preprostega protokola izliv – odgovor, kjer se avtenticirata oba sogovornika

Žal je tak protokol ranljiv za napad z zrcaljenjem (angl. *reflection attack*), kot ga prikazuje slika 7.14. Predpostavljamo, da vsak od sogovornikov podpira več sočasnih sej z različnimi sogovorniki, kar zagotovo velja vsaj za večino strežnikov. Napadalec se predstavi kot Ana, Borutu pošlje izliv in Borut ubogljivo vrne napadalcu kriptiran izliv napadalca ter še svoj izliv.

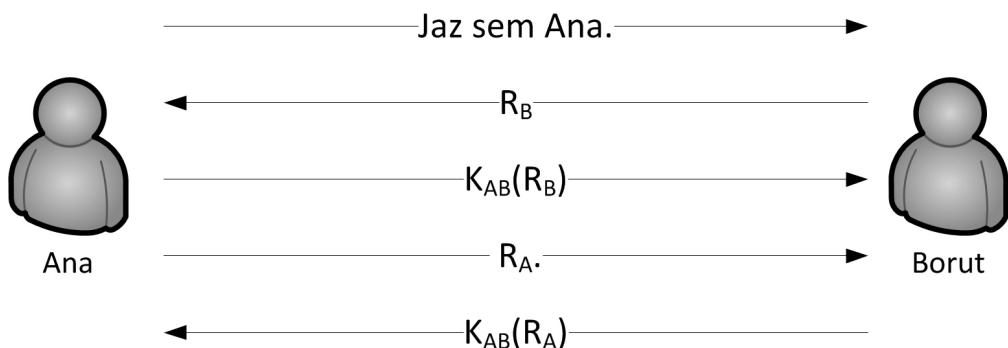
Nato napadalec odpre še eno sejo z Borutom (na sliki označeno s sivo puščico in zvezdico), ponovno se predstavi kot Ana in Borutu pošlje svoj izliv. Vendar sedaj za svoj izliv vzame  $R_B$ , torej natanko tisto število, ki ga je Borut poslal kot izliv napadalcu, da dokaže, če je res Ana. Borut zopet ubogljivo deluje po protokolu in vrne kriptiran izliv. Napadalca sedaj ta seja več ne zanima in jo opusti, shrani le kriptiran izliv in ga v naslednjem koraku uporabi v začetni



Slika 7.14 Napad z zrcaljenjem

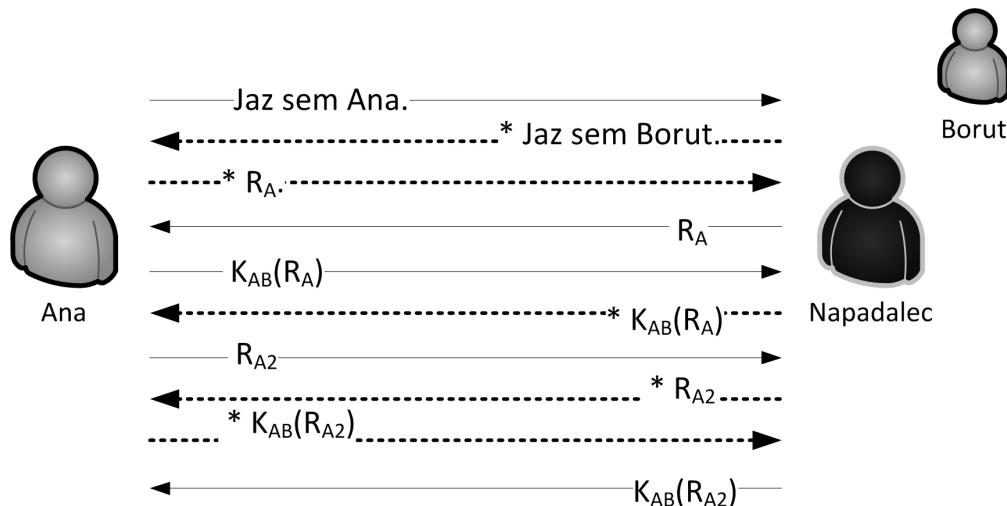
(zvezni) seji in tako dokaze svojo lažno identiteto. Borut je po prejemu  $K_{AB}(R_B)$  prepričan, da komunicira z Ano.

Opisani protokol je slab že zato, ker stran, ki vzpostavlja komunikacijo, prva zahteva avtentikacijo od druge strani. To je voda na mlin napadalcu za izvajanje napada z zrcaljenjem. Kaj pa, če bi se moral najprej avtenticirati tisti, ki vzpostavlja komunikacijo, kot prikazuje slika 7.15?



Slika 7.15 Preprost protokol izziv-odgovor, kjer se najprej avtenticira tisti, ki vzpostavlja komunikacijo

Ali je ta protokol varnejši? Gotovo ni možen popolnoma enak napad z zrcaljenjem, ki bi ga v celoti narekoval napadalec. Vendar lahko napadalec še kljub vsemu potrpežljivo posluša in čaka, da eden od legalnih uporabnikov začne vzpostavljati komunikacijo. Ko se to zgoditi, je napadalec na konju in lahko izvede napad, kot kaže slika 7.16. Tudi tokrat sta potrebni dve seji in na sliki jih lahko ločimo po slogu puščic – imenujmo ju zvezna in črtkana seja. Sporočila črtkane seje so zaradi boljšega razločevanja med sejama označena še z zvezdico.



**Slika 7.16** Tudi kadar napadalec ni pobudnik vzpostavljanja komunikacije je možen napad z zrcaljenjem

Napadalec prestreže in zadrži Anino sporočilo, ki je namenjeno Borutu, vendar ji namesto Boruta še ne odgovori. Pač pa odpre novo sejo z Ano in počaka, da mu Ana pošlje svoj izziv. Tega ji potem podtakne v prvi (zvezni) seji, saj ve, da mu bo Ana v naslednjem sporočilu vrnila pravilno kriptiran izziv. Tega napadalec nato v črtkani seji v naslednjem koraku uporabi za dokaz svoje lažne identitete (Borut). V zvezni seji je zdaj Ana na vrsti, da pošlje sogovorniku svoj izziv ( $R_{A2}$ ). Ker ga napadalec ne zna pravilno kriptirati, ji tudi ta izziv vrne kot svoj izziv v črtkani seji, Ana pa mu takoj vrne kriptiranega s ključem  $R_{AB}$ , ki predstavlja skupno skrivnost Ane in Boruta. Napadalcu preostane le še, da kriptirani izziv vrne Ani v zvezni seji in to je zadnji korak avtentikacije. Napadalec ima sedaj odprti dve seji z Ano in v obeh se je uspešno avtenticiral kot Borut. Niti Ana niti Borut ne dvomita, da bi lahko bilo karkoli narobe.

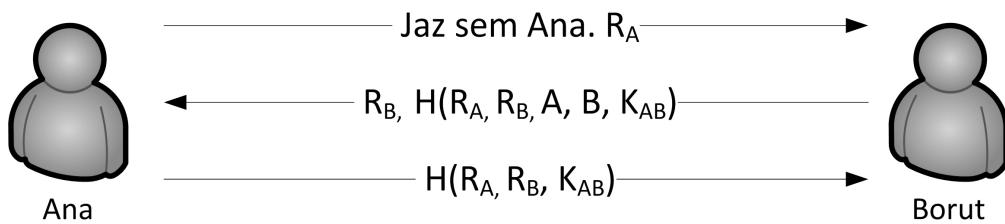
Dejstvo je, da je varen protokol izziv-odgovor težko zgraditi. V sodobnih implementacijah tega protokola za večjo stopnjo varnosti upoštevamo še naslednja štiri zlata pravila:

1. Kdor vzpostavlja komunikacijo, naj tudi prvi dokaže svojo identiteto. To pravilo napadalcu le malo oteži delo, saj ne more sprožiti napada v poljubnem trenutku, ampak mora čakati, da bodoča žrtev sama začne komunikacijo.
2. Za dokaz identitete naj sogovornika uporablja različna ključa. Še vedno gre za simetrično kriptografijo, vendar ob vzpostavljanju skupne skrivnosti določimo dve vrednosti ključa, ki ju poznata obe strani, in se dogovorimo, kdo bo uporabljal katerega. To pravilo preprečuje napad z zrcaljenjem. Napadalec lahko prejeti izziv sicer vrne v drugi seji istemu procesu, a ta

bo izziv kriptiral z napačnim ključem in napadalec si s tem ne bo mogel pomagati.

3. Izziva naj bosta iz različnih množic, na primer en sogovornik naj uporablja soda, drugi pa liha števila. To pravilo deluje podobno kot prejšnje – preprečuje napad z zrcaljenjem, saj napadalec lahko prejeti izziv sicer vrne v drugi seji istemu procesu, ta ga vrne kriptiranega, a napadalcu to pri avtentikaciji ne bo pomagalo, ker izziv ne bo iz prave množice.
4. Informacija iz ene seje naj bo nekoristna v drugi (pomagamo si z identifikatorji seje, časovnim žigosanjem in podobno). To pravilo preprečuje napade s posneto sejo. Če bo napadalec poslal sporočila iz neke stare seje, bo prejemnik to lahko ugotovil in jih bo zavrgel.

Varnejši protokol izziv-odgovor pa dobimo, če namesto enkripcije uporabimo zgoščevalne funkcije. Še vedno morata sogovornika vnaprej vzpostaviti skupno skrivnost, vendar v tem primeru ni uporabljena kot enkripcijski ključ. Osnovni protokol prikazuje slika 7.17.



**Slika 7.17** Protokol izziv-odgovor z zgoščevalnimi funkcijami

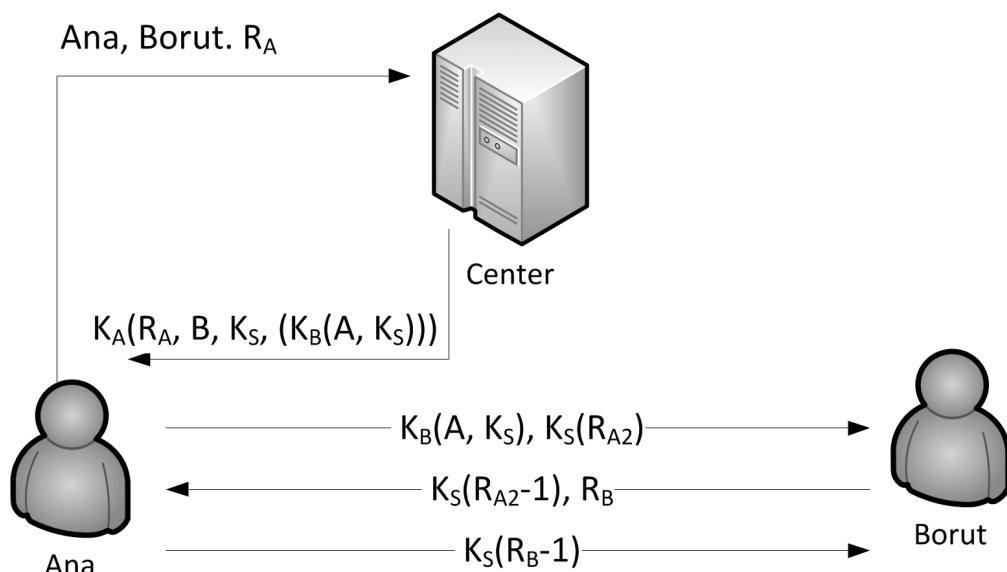
Sedaj Ana pošlje Borutu nekriptiran izziv, ki ga lahko brez škode prestreže tudi napadalec. Borut ji vrne zgoščeno vrednost peterke podatkov: njen izziv, njegov izziv, njena identiteta, njegova identiteta in skupna skrivnost. Poleg tega ji pošlje še nekriptiran in nezgoščen lastni izziv. Ana pozna vse te podatke in lahko preveri, ali je poslana zgoščena vrednost pravilna. Ko se o tem prepriča, tudi sama dokaže svojo identiteto tako, da Borutu pošlje zgoščeno vrednost treh podatkov: obeh izzivov in skupne skrivnosti.

Napad z zrcaljenjem tu ni možen, saj napadalec na nobenem koraku ne more nobenemu od sogovornikov "podtakniti" izziva za izračun zgoščene vrednosti. Na vsakem koraku mu za izvedbo napada manjka kak podatek.

### Simetrična avtentikacija z osrednjo avtoritetom

Glavna težava protokolov izziv-odgovor je vzpostavljanje skupne skrivnosti. Ker imamo lahko na tisoče potencialnih sogovornikov, postane upravljanje s ključi naporno. Vzpostavljanje ključa prek ne-varnega interneta je ranljivo, protokol

Diffie-Hellman ni dovolj varen. Zato je enostavnejše, če upravljanje s ključi prepustimo osrednji avtoriteti, ki ji zaupamo vsi potencialni sogovorniki in ki pozna skrivnosti (ključe) vsakega od sogovornikov. Če želi nekdo komunicirati v takem sistemu, se mora najprej prijaviti pri takšni avtoriteti – centru za distribucijo ključev, ki mu dodeli lasten ključ za avtentikacijo. Uporablja se simetrična kriptografija. Protokol, ki nosi ime po svojih predlagateljih (Needham-Schroeder), kaže, kako Ana vzpostavi komunikacijo z Borutom, pri čemer vedno vključuje tudi center, kot kaže slika 7.18. Takšno avtentikacijo uporablja tudi Kerberos. Kerberos je ime za avtentikacijski protokol in avtentikacijski strežnik (implementacijo) oziroma center za distribucijo ključev po principu Needham-Schroeder.



**Slika 7.18** Protokol Needham-Schroeder in uporaba simetrične kriptografije in osrednje avtoritete za hranjenje in preverjanje ključev za avtentikacijo

Odjemalec Ana želi dostop do strežnika Boruta. Ana se najprej dogovori za kriptiranje seje s strežnikom Kerberos – uporabi se Anin simetrični ključ, ki ga poznata le Ana in Kerberos. Strežnik Kerberos pošlje Ani v kriptiranem sporočilu njen izziv, Borutovo identiteto, sejni ključ za komunikacijo z Borutom ter vstopnico – sporočilo, s katerim se bo avtenticirala Borutu. To sporočilo je kriptirano z Borutovim simetričnim ključem, ki ga poznata le Borut in Kerberos. V njem pa sta zapisana Anina identiteta in sejni ključ za komunikacijo med Ano in Borutom.

Ana pošlje Borutu vstopnico, ki je še vedno kriptirana z Borutovim simetričnim ključem (seveda, saj je Ana ne zna dekriptirati). Poleg tega mu pošlje tudi izziv, kriptiran s sejnim ključem. Borut dekriptira vstopnico s svojim simetričnim ključem in s tem dokaže svojo identiteto. V njej najde sejni ključ, s katerim dekriptira še

drugo sporočilo z izzivom. Izzivu odšteje 1, ga ponovno kriptira s sejnim ključem in vrne Ani. Ana se iz prejetega sporočila lahko prepriča, da je Borut dekriptiral vstopnico, saj očitno razpolaga s sejnim ključem. Torej verjame, da komunicira s pravim Borutom. Prepričati pa je potrebno še Boruta, zato mu Ana vrne njegov izziv, a ga kriptira s sejnim ključem. Ko Borut vidi, da tudi Ana razpolaga s sejnim ključem, ve, da je moral za to uporabiti Anin simetrični ključ, torej je tudi ona dokazala svojo identiteto.

Takšen sistem je relativno varen in napad je možen le na koraku, ko Ana pošle Borutu vstopnico in kriptiran izziv. To je napad s posneto sejo (angl. *replay attack*), kjer napadalec shranjuje vsa poslana sporočila določene seje, čeprav nima ključa. Če ima napadalec shranjeno oz. posneto neko sejo iz preteklosti in se hkrati še dokoplje do sejnega ključa te seje (nič zato, če se je ta že zaključila), lahko to staro sejo ponovno odigra – Borutu ponovno pošle vstopnico in izziv in Borut ne more vedeti, da ne gre za vzpostavljanje čisto nove seje. Tako lahko napadalec na primer povzroči ponovno plačilo računa iz vaših sredstev in se s tem okoristi, vas pa oškoduje. Vendar pa napad ni enostaven (težavno je predvsem razbijanje starega sejnega ključa) in je morda manj verjeten, kot ostali do sedaj opisani napadi. Sistemi Kerberos se danes pogosto uporabljajo in jih štejemo za varne, zaradi opisane možnosti napada pa Kerberosa ne uporabljajo na primer banke in podobne institucije z visokimi varnostnimi zahtevami.

Kako pa napad preprečujemo? Bolj kot omejimo čas veljavnosti vstopnice, bolj napadalcu zvežemo roke, saj zmanjšujemo časovni okvir, ki ga ima na razpolago, da pride do sejnega ključa. Kerberos tako na primer v samo vstopnico vključuje tudi njen čas veljavnosti, ki je relativno kratek (privzeto je ta čas navadno pol minute). Časovno okno, v katerem mora napadalec pridobiti sejni ključ, je namreč dolgo natanko toliko, kot je čas veljavnosti vstopnice.

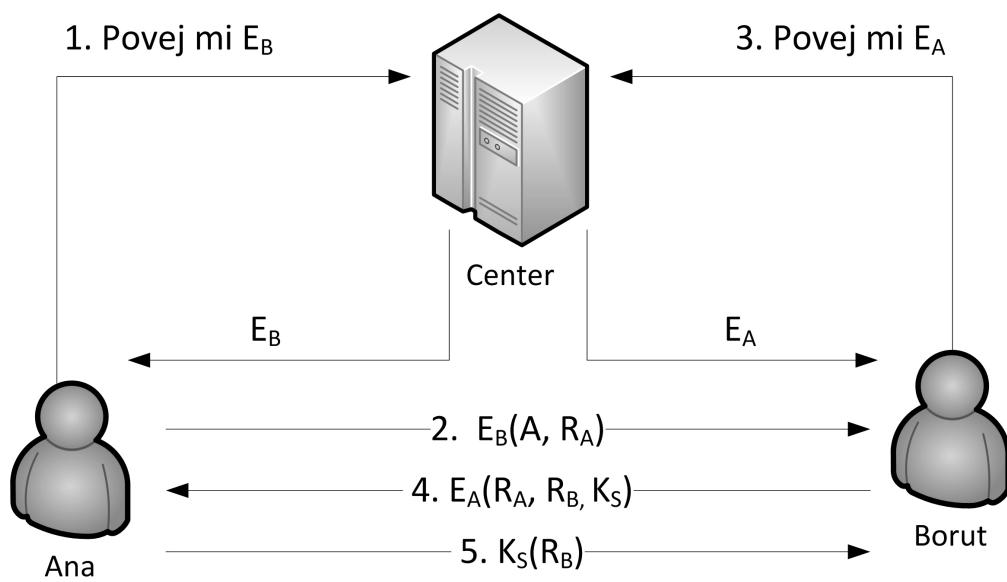
### **Asimetrična (PKI) avtentikacija z osrednjo avtoritetom**

V varnostno zelo občutljivih aplikacijah, na primer bančnih, davčnih in podobno, želimo varnostna tveganja zmanjšati na minimum in se nam Needham-Schroederjev protokol ne zdi dovolj varen, zato uporabimo avtentikacijo z osrednjo avtoritetom in asimetrično kriptografijo. Standardiziran model takega sistema imenujmo infrastruktura javnih ključev oziroma PKI (angl. *Public Key Infrastructure*). Kadar se uporablja infrastruktura javnih ključev (PKI), ravno tako kot v primeru s simetrično kriptografijo, predpostavljamo obstoj osrednje avtoritete, ki ji privzeto zaupamo: digitalnega overitelja, notarja ali certifikatne agencije (CA).

Tovrstno avtentikacijo imamo danes za najbolj varno, težava pa je, da za uporabnika ni tako zelo prijazna, kot bi si žeeli. Uporabnik se mora pred uporabo prijaviti osrednji avtoriteti in ta poskrbi za vzpostavitev uporabnikovega javnega in tajnega ključa za asimetrično kriptografijo, navadno je to RSA, lahko pa bi uporabljali tudi kriptografijo z eliptičnimi krivuljami. Vsak uporabnik ima torej par

ključev – tajnega na svojem računalniku in javnega pri osrednji avtoriteti (in lahko še kje drugje).

Osnovni avtentikacijski postopek prikazuje slika 7.19. Če želi Ana dostopati do virov na Borutovem strežniku, zaprosi najprej centralo za Borutov javni ključ. Ko ga dobi, kriptira s tem ključem svojo identiteto in izziv, ki ga pošlje Borutu. Borut s svojim tajnim ključem sporočilo dekriptira (ker ima tajni ključ izključno samo on, tega sporočila morebitni napadalec ne more prebrati). Ugotovi, da želi komunicirati z njim Ana, in zaprosi centralo za Anin javni ključ. Z Aninim javnim ključem nato kriptira sporočilo, ki vsebuje tri vrednosti: Anin izziv, svoj izziv in sejni ključ, ki ga generira sam. Sejni ključ je tipično simetričen in se bo po zaključeni obojestranski avtentikaciji uporabljal za nadaljnje kriptiranje seje, saj je asimetrična kriptografija za takšno masovno uporabo prepočasna. Ko Ana sprejme Borutovo sporočilo, ga dekriptira s svojim tajnim ključem in ugotovi, da ji je Borut vrnil izziv, ki mu ga je poslala v prejšnjem koraku. To pomeni, da ga je uspešno dekriptiral. Ker se dekripcijo lahko opravi le z Borutovim tajnim ključem in ker tega ključa nima nihče drug kot Borut, je Ana sedaj prepričana, da res komunicira s pravim Borutom – Borut je torej avtenticiran. Ana nato vrne Borutu njegov izziv, kriptiran s sejnim ključem in Borut ga po prejemu s sejnim ključem tudi dekriptira. Ko ugotovi, da je v sporočilu shranjen pravi izziv, ve tudi, da ga je poslala prava Ana. Samo Ana je namreč lahko s svojim tajnim ključem, ki ga nima nihče drug kot ona, dekriptirala prej poslano sporočilo in iz njega razbrala sejni ključ in Borutov izziv.



**Slika 7.19** PKI – infrastruktura javnih ključev in uporaba asimetrične kriptografije in osrednje avtoritete za hranjenje in preverjanje ključev za avtentikacijo

## 7.7 Integriteta komunikacije

Spomnimo se, da za zagotavljanje integritete komunikacije uporabljamo zgoščevalne kriptografske funkcije (angl. *hash functions*), ki poljubno sporočilo preslikajo v t. i. zgoščeno vrednost, običajno fiksne dolžine. Za dobre zgoščevalne funkcije velja, da so:

- preprosto izračunljive,
- da iz zgoščene vrednosti ne moremo ugotoviti vrednosti izvornega sporočila,
- da je zelo težko najti dva čistopisa, ki se preslikata v isto zgoščeno vrednost in
- da daje njihov zapis vtis naključnega zaporedja znakov.

V nadaljevanju si poglejmo nekaj pristopov za zagotavljanje integritete komunikacije.

### 7.7.1 Pripenjanje zgoščene vrednosti

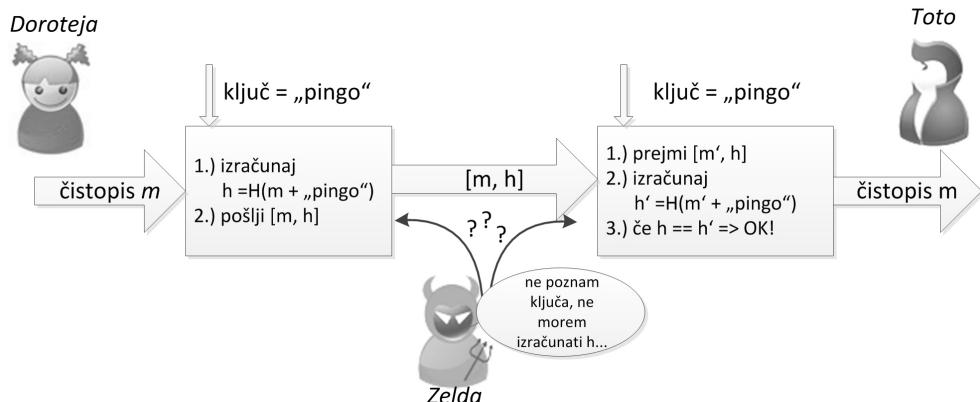
Sporočilu  $m$  pri pošiljanju lahko pripnemo še njegovo zgoščeno vrednost  $H(m)$ . Prejemnik na svoji strani prejme sporočilo  $m'$  (originalno sporočilo  $m$  je napadalec namreč lahko spremenil), na podlagi katerega izračuna zgoščeno vrednost  $H(m')$  z uporabo iste zgoščevalne funkcije, kot jo je uporabil pošiljatelj. V primeru, če je bilo sporočilo spremenjeno, torej če  $m \neq m'$ , bo torej veljalo tudi  $H(m) \neq H(m')$ , kar da prejemniku vedeti, da je nekdo posegel v sporočilo. Tak sistem zagotavljanja integritete je učinkovit, ne obnese pa se, kadar morebitni ponarejevalec sporočil pozna zgoščevalno funkcijo in lahko ob zamenjavi sporočila  $m$  za  $m'$  sam izračuna pravilno zgoščeno vrednost sporočila ter jo pripne ponarejenemu sporočilu.

### 7.7.2 Zgoščanje z avtentikacijskim ključem

Pomanjkljivosti zgornje metode se lahko izognemo, če pošiljatelj in prejemnik uporabljata t.i. *avtentikacijski* ali *tajni* (angl. *secret*) *ključ*. Avtentikacijski ključ  $s$  je vrednost, za katero se dogovorita pošiljatelj in prejemnik in jo pri zgoščanju pripenjata originalnemu sporočilu. Pošiljatelj bo torej sporočilu  $m$  pripel zgoščeno vrednost  $H(m + s)$ , prejemnik bo pa edini, ki bo vedel, da mora nedotaknjenost prejeta sporočila  $m'$  preveriti z izračunom  $H(m' + s)$ , saj napadalec vrednosti ključa  $s$  ne pozna. Na ta način lahko napadalec sicer originalno sporočilo ponaredi in zamenja s svojim, ne more pa mu pripeti veljavne zgoščene vrednosti, kar bo prejemnik zaznal in sporočilo zavrgel.

Primer zgoščanja z avtentikacijskim ključem je prikazan na sliki 7.20. Na sliki Doroteja in Toto uporabljata ključ "pingo", ki je znan le njima. Ker ga Zelda ne

pozna, ne more ponarediti pripete zgoščene vrednosti  $h$  tako, da bi se ujemala s sporočilom  $m$ .

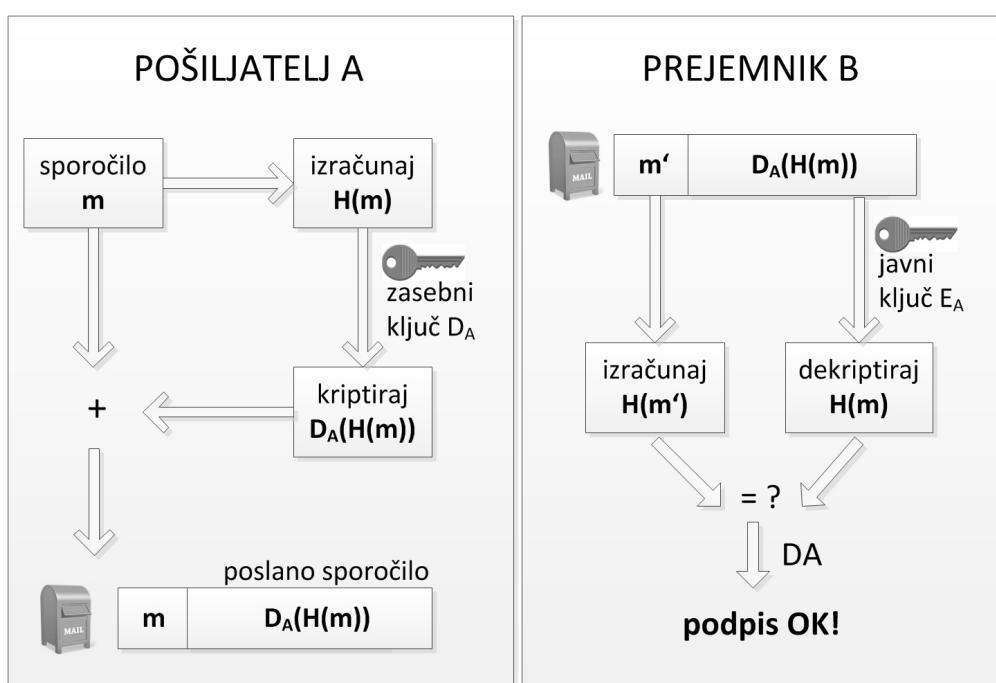


**Slika 7.20** Primer zgoščanja z avtentikacijskim ključem

Opozorimo, da ima metoda težavo z distribucijo avtentikacijskega ključa, saj se morata pošiljatelj in prejemnik pred pošiljanjem sporočila zanj najprej dogovoriti.

### 7.7.3 Elektronski podpis

V praksi najbolj razširjena in učinkovita različica preverjanja **integritete in avtentikacije** pošiljatelja uporablja kriptografijo RSA in postopek, ki ga imenujemo **digitalni podpis**. Spomnimo se, da ima vsak uporabnik kriptografije RSA dva ključa: javni enkripcionski ključ  $E$  in zasebni dekripcionski ključ  $D$ . Pošiljatelj A zagotovi integriteto in svojo avtentikacijo sporočila  $m$  tako, da sporočilu pripne zgoščeno vrednost sporočila, kriptirano s svojim zasebnim ključem  $D_A(H(m))$ . Prejemnik, ki lahko z javnim ključem pošiljatelja odkriptira prejeto kriptiptirano zgoščeno vrednost, izračuna  $E_A(D_A(H(m))) = H(m)$ . Na podlagi nekriptiranega sporočila  $m'$ , ki ga je tudi prejel, lahko izračuna  $H(m')$  in preveri, ali velja ujemanje dekriptiranega in izračunanega  $H(m) = H(m')$ , kot je prikazano na sliki 7.21.



**Slika 7.21** Postopek preverjanja elektronskega podpisa



# Literatura

- [1] Cisco Networking Academy. *Routing Protocols Companion Guide*. Cisco Press, 2014.
- [2] F.K. Gürkaynak. *GALS system design: side channel attack secure cryptographic accelerators*. ETH, 2006.
- [3] V. Klima. Finding md5 collisions – a toy for a notebook. Cryptology ePrint Archive Report 2005/075, 5 March 2005, revised 8 March 2005.
- [4] F. Kurose and K. Ross. *Computer Networking: A top-down approach*. Pearson, 6. izdaja, 2013.
- [5] A. Tanenbaum and D. Wetherall. *Computer networks*. Pearson, 5. izdaja, 2011.
- [6] T. Vidmar. *Informacijsko-komunikacijski sistem*. Pasadena, Ljubljana, 2002.
- [7] Wikipedia. Bifid cipher. [http://en.wikipedia.org/wiki/Bifid\\_cipher](http://en.wikipedia.org/wiki/Bifid_cipher).
- [8] Wikipedia. Caesar cipher. [http://en.wikipedia.org/wiki/Caesar\\_cipher](http://en.wikipedia.org/wiki/Caesar_cipher).
- [9] Wikipedia. Great cipher. [http://en.wikipedia.org/wiki/Great\\_Cipher](http://en.wikipedia.org/wiki/Great_Cipher).
- [10] Wikipedia. OSI model. [http://en.wikipedia.org/wiki/OSI\\_model](http://en.wikipedia.org/wiki/OSI_model).
- [11] Wikipedia. TCP IP model. [http://en.wikipedia.org/wiki/TCP/IP\\_model](http://en.wikipedia.org/wiki/TCP/IP_model).
- [12] Wikipedia. Vigenère cipher. [http://en.wikipedia.org/wiki/Vigen%C3%A8re\\_cipher](http://en.wikipedia.org/wiki/Vigen%C3%A8re_cipher).