Modeliranje računalniških omrežij

Miha Mraz in Miha Moškon

12.12.2016

Kazalo

Poglavje 1

Uvod

1.1 Osnovni pojmi modeliranja računalniških omrežij

Računalniška omrežja so v zadnjih dveh desetletjih izrazito napredovala tako z vidika dviga *zmogljivosti*, kot tudi z vidika dviga *zanesljivosti* delovanja. Pod pojmom zmogljivosti smatramo predvsem stalno povečevanje hitrosti prenosa in zagotavljanje prihajanja paketov v pravilnem vrstnem redu, pod pojmom zanesljivosti pa vse večjo *dosegljivost* delujočega omrežja. Slednjo merimo z odstotkom delujočega stanja omrežja, gledano preko daljšega časovnega obdobja (npr. 99,999%). K obema napredkoma so doprinesle tako izboljšane tehnologije prenosa po medijih (npr. optične tehnologije), kot tudi vsebinske (logične) izboljšave protokolov in njihove vse hitrejše izvedbe.

Samo omrežje je za uporabnika postalo do neke mere "nevidno", saj uporabnikovo delo temelji na storitvah, ki mu jih slednje ponuja, v tem kontekstu pa je tako pozoren le na *dosegljivost teh storitev*. Dosegljivosti storitev ne smemo enačiti z *dosegljivostjo omrežja*, ki je načeloma vsaj tako velika kot dosegljivost storitve, praviloma pa večja, saj kljub normalnemu delovanju omrežja posamezna storitev ni nujno dosegljiva (npr. nedelovanje strežnika elektronske pošte, bančnega avtomata, spletnega strežnika itd.). Uporabnika tehnične značilnosti same izvedbe priklopa v omrežje ne zanimajo, je pa pozoren na predhodno omenjena faktorja zmogljivosti in zanesljivosti delovanja. Prvega uporabnik ocenjuje skozi hitrost pretoka podatkov do njega (angl. *download*) in od njega (angl. *upload*). Obe hitrosti merimo v bps (angl. *bits per second*). Zanesljivost delovanja omrežja uporabnik ocenjuje s predhodno navedeno dosegljivostjo omrežja in s številom napak, do katerih pride pri prenosu paketov.

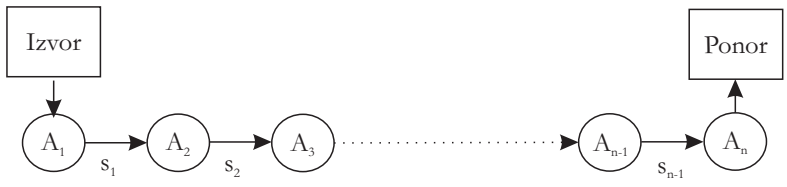
V pričujočem delu obravnavamo predvsem metode modeliranja pretoka podatkov po omrežjih. Slednji je lahko z vidika uporabnika *običajen* (hitrost prenosa je približno enaka deklarirani hitrosti, ki jo uporabnik pričakuje), ali pa *degradiran* (okrnjen), ker je v omrežju preveč zahtev in posledično na posameznih točkah predvidene poti prihaja do zastojev zaradi zasičenosti določenih

sestavnih delov omrežja. S samimi odpovedmi omrežja se v pričujočem delu ne bomo ukvarjali.

Z vidika modeliranja in realizacije komunikacijskih povezav bomo pod pojmom *komunikacijske poti* smatrali naslednje sestavne gradnike:

- *izvor zahtev* (paketov, podatkov, itd.),
- *pot po omrežju* in,
- *ponor zahtev*.

Pod pojmom izvora smatramo napravo, ki je zahtevo poslala v omrežje z znanim naslovnikom (ponorom), pod pojmom poti po omrežju *zaporedje fizičnih prenosnih medijev* in *aktivnih omrežnih komponent* (npr. usmerjevalnikov), pod pojmom ponora zahtev pa napravo, ki naj bi jo zahteva dosegla. Na sliki 1.1 je predstavljena tipična komunikacijska pot, ki je sestavljena iz prenosnih fizičnih medijev (s_1 do s_{n-1}) in omrežnih komponent (A_1 do A_n). Poudariti je potrebno, da se pot po omrežju med izvorom in ponorom lahko glede na obremenitve posameznih fizičnih prenosnih medijev in aktivnih omrežnih komponent skozi čas spreminja, za kar poskrbi integracija inteligentnih protokolov v posamezne aktivne omrežne komponente.



Slika 1.1: Grafični prikaz komunikacijske poti med ponorom in izvorom.

Na vsaki aktivni omrežni komponenti v omrežju je potrebno zahtevo preusmeriti na ustrezno destinacijo (preko izbranega fizičnega prenosnega medija proti izbrani omrežni komponenti), kar naj bi zahtevo v časovnem smislu poteka poti približalo ponoru (naslovniku). Za slednje so zadolženi *komunikacijski protokoli*.

1.2 Zahteve, strežniki, čakalne vrste

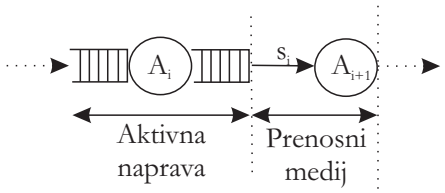
Osnovni pojmi modeliranja računalniških omrežij v domeni pričujočega dela so *zahteve*, *strežniki* in *čakalne vrste*. Njihovi pomeni so sledeči:

- pod pojmom zahteve smatramo nerazčlenljivo entiteto, ki potuje po računalniškem omrežju; običajno je to paket, v katerega so vpeti vsebinski in dodani kontrolni podatki; v splošno namenskih modelirno simulacijskih programskih orodjih je zahteva poimenovana tudi s pojmom *transakcija*, ali *entiteta*;

- pod pojmom strežnika smatramo posamezen segment omrežja, ki ga zahteva (paket) za določen čas zasede, da bi bila postrežena; strežniki so lahko tako fizični prenosni mediji (zahteve zavzamejo določene časovne rezine v odvisnosti od propustnosti medija), kot tudi aktivne komponente, na katerih poteka procesiranje in pošiljanje zahtev v druge segmente omrežja; v splošno namenskih modelirno simulacijskih programskih orodjih je strežnik pogosto poimenovan s pojmom *resurs*,

- pod pojmom čakalne vrste smatramo posamezen vmesnik (angl. *buffer*), v katerem zahteva čaka, da bo *postrežena* (obdelana ali sprocesirana) na strežniku; na področju računalniških omrežij do porajanja čakalne vrste lahko pride tako pred vsako aktivno komponento (čakanje na izvedbo preusmeritve poti), kot tudi na izstopu iz aktivne komponente (čakanje na prosto časovno rezino za odpošiljanje po prenosnem mediju).

V kontekstu povedanega bi posamezno aktivno komponento in fizični prenosni medij iz slike 1.1 lahko opcijsko dopolnili s čakalnimi vrstami, kot je predstavljeno na sliki 1.2.



Slika 1.2: Čakalni vrsti pred in za aktivno omrežno komponento.

1.3 Modeliranje in simulacije toka paketov v računalniških omrežjih

Primarni namen pričujočega dela je pregled metod za postavitev modelov računalniških omrežij, ki jih uporabljamo za izvajanje simulacij dinamike paketov v omrežjih. Glede na simulacijske rezultate modelu po potrebi dodajamo strežne komponente (fizične prenosne medije in aktivne omrežne komponente), s čimer praviloma povečamo hitrost delovanja, lahko pa strežne komponente tudi odvzemamo, s čimer dinamiko praviloma upočasnjujemo. Prva aktivnost omrežju ceno zvišuje, druga pa mu jo znižuje. Izboljšanje dinamike lahko dosežemo tudi s spreminjanjem performans (zmogljivosti) posameznih strežnih komponent (npr. spreminjanje procesnih zmožnosti, velikosti vmesnikov itd.)

Glavni cilji modeliranja računalniških omrežij so sledeči:

- iskanje kompromisa med ceno in zmogljivostjo omrežja, ki je za končnega uporabnika in (ali) upravljalca - ponudnika še sprejemljiva,

- določanje odzivanja omrežja pri nepredvidenih dogodkih,
- izvajanje IF-THEN analize,
- omogočanje natančnega uglaševanja omrežja (angl. *fine tunning*), itd.

Osnovni pojmi na področju modeliranja računalniških omrežij so sledeči:

- *opazovani omrežni sistem*: pod tem pojmom smatramo realni (že obstoječi) omrežni sistem ali sistem, ki je šele v fazi načrtovanja (omrežni sistem še ni realiziran),
- *komponente sistema*: pod tem pojmom smatramo množico vseh sestavnih delov omrežja (prenosne medije in aktivne komponente), ki sestavljajo komunikacijske poti,
- *stanje sistema*: pod tem pojmom smatramo množico vseh tipov zahtev, njihovo število in njihove pozicije v sistemu,
- *model sistema*: pod tem pojmom smatramo opis komponent sistema in formalni zapis relacij med njimi,
- *simulacija v omrežnem sistemu*: pod tem pojmom smatramo postopke navideznega izvajanja strežbe (izvajanje modela), ki nam dajo opisne rezultate o dinamiki (pretoku zahtev); model lahko prožimo z različnimi začetnimi pogoji (npr. uporaba različnih komponent, različnih topoloških konfiguracij komponent sistema ter različnih vhodnih bremen - zahtev),
- *verifikacija modela*: pod tem pojmom smatramo korake preverjanja konsistentnosti postavljenega modela in njegovih simulacijskih rezultatov v primerjavi z delovanjem referenčnega sistema; s tem vzpostavimo zaupanje v konsistentnost modela (potrditev modela),
- *simulacijska hipoteza*: pod tem pojmom smatramo predpostavko o simulacijskih rezultatih, iz katere izhajamo ob vsakem proženju simulacije ob vnaprej določenih robnih pogojih (zgradbi in ključnih parametrih modela); če bi v modelu npr. povečali prepustnost posameznih aktivnih komponent, bi bila simulacijska hipoteza predpostavka, da se bo pretok zahtev skozi omrežje ustrezno povečal; v mislih imamo torej predpostavke, vezane na spremembe značilnosti posameznih komponent sistema.

Postavljanje modelov sistemov in proženje simulacij na osnovi tovrstnih modelov, je prav gotovo ena od disciplin računalništva, ki je z vidika masovne uporabe zapostavljena. Na mnogih področjih, kjer je eksperimentiranje v realnem okolju lahko prenevarno, cenovno predrago ali časovno potratno, je pogosto najboljši način vzpostavitev ustreznih modelov in izvajanje simulacij na njihovi osnovi. Tako lahko postavljamo modele na najrazličnejših področjih računalništva pa tudi drugje. Ta so na primer modeli pretoka cestnega ali železniškega prometa, poslovanja strežno orientiranih sistemov kot so banke, trgovine, restavracije itd.

1.4 Namen in pregled dela

Pričujoče delo je namenjeno slušateljem predmeta *Modeliranje računalniških omrežij* izbirnega modula Računalniška omrežja, ki se je začel predavati jeseni leta 2011 v 3. letniku bolonjskega univerzitetnega študija 1.stopnje na Fakulteti za računalništvo in informatiko Univerze v Ljubljani. Avtorja se bralcem že vnaprej opravičujeva za vse potencialne napake, ki se v takšno delo prikradejo navkljub mnogim korekcijam. Vse napake bodo sproti objavljane na spletni strani predmeta, o čemer bodo obveščeni slušatelji vsake generacije posebej.

V drugem poglavju pričujočega dela predstavimo osnovne pojme modeliranja in simulacij, v tretjem pa osnove teorije strežbe. Kakršnokoli računalniško omrežje in promet, ki se na njem odvija, lahko smatramo kot množico zahtev, strežnikov (resursov) in pa eventuelnih čakalnih vrst, ki upočasnjujejo delovanje omrežja in s tem uporabnika navajajo k razmišljanju o možni izbiri drugega ponudnika omrežnih storitev (angl. *internet provider*). V četrtem poglavju predstavimo modelirno okolje Petrijevih mrež. V petem poglavju predstavimo nagradnjo Petrijevih mrež, in sicer s tako imenovanimi barvnimi Petrijevimi mrežami (angl. *coloured Petri nets*), ki so bile razvite prav za modeliranje računalniških protokolov. V šestem poglavju se posvetimo pojmu latence v računalniških omrežjih, v sedmem in hkrati zadnjem poglavju pa predstavimo javno dostopno modelirno razvojno okolje OMNet++, ki se je začelo pri vajah pričujočega predmeta uporabljati v štud. l. 2011/2012.

Poglavje 2

Uvod v modeliranje in simulacije

2.1 Osnovni pojmi

Pričujoče poglavje je deloma povzeto po viru [1] in je posvečeno razlagi osnovnih pojmov, s katerimi se srečamo pri modeliranju in simulaciji. *Modeliranje* je definirano kot proces, ki nas pripelje do *modela* realnega oziroma že obstoječega sistema ali pa do modela sistema, ki ga želimo z njegovo pomočjo načrtovati. Vzpostavljeni model je lahko verna slika sistema ali pa še pogosteje njegova posplošitev. Glavni namen modela je, da nam omogoča hitre spremembe strukture (arhitekture in parametrov) opazovanega sistema in napove posledice, ki jih te spremembe vpeljejo v dinamiko sistema. V splošnem modeli temeljijo na matematičnih relacijah, v večini primerov pa jih postavljamo s pomočjo simulacijskih programskih orodij, ki so lahko splošno namenska (npr. GPSS, SimProcess itd.), ali specializirana za posamezno področje dela. Tipičen primer slednjih je javno dostopno razvojno okolje OMNet++, ki se uporablja za modeliranje računalniških omrežij.

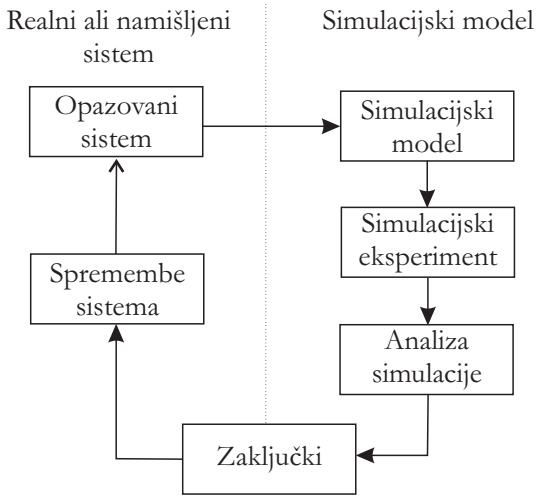
Po postavitvi modela sledi njegova *verifikacija*. Izvedemo jo tako, da modelu na vhodu dostavimo podatke, za katere vemo, kako se nanje odzove realni (že obstoječi) sistem oziroma predpostavljamo želene odzive še neobstoječega sistema. Če model izkazuje vnaprej določeno stopnjo ujemanja z realnim oziroma želenim odzivom, je verifikacija uspešna, zato mu lahko (bolj ali manj) zaupamo.

Pod pojmom *simulacije* smatramo „izvajanje“ modela, ki nam predstavi dinamiko v opazovanem sistemu. Splošneje bi lahko zapisali, da je simulacija po eni plati orodje za evaluacijo zmogljivosti (angl. *performance*) opazovanega sistema pri različnih konfiguracijah in različnih sestavnih delih sistema, po drugi plati pa jo lahko smatramo kot orodje, ki nam omogoči analizo dinamike in odziva sistema. Simulacija namreč lahko pokaže, da dinamika v sistemu ni ustrezna in se izkaže kot nepravilna. Odtod si lahko postavimo odgovor na vprašanje, čemu uporabljati modeliranje in simulacije. Uporabljamo ju za *analizo*

pravilnosti delovanja in *analizo zmogljivosti delovanja* opazovanega sistema. V kontekstu računalniških omrežij bi bil primer prve analize preverjanje *pravilnosti delovanja novo zasnovanih protokolov*, primer druge analize pa iskanje *ozkih grl* sistemov, ki so v večini primerov vzroki za upočasnjevanje prometa na posamezni komunikacijski poti. Pod pojmom ozkega grla smatramo strežni gradnik v sistemu, pred katerim prihaja zaradi njegove premajhne strežne zmogljivosti do prekomernega polnjenja čakalnih vrst in s tem posredno do prekomernega čakanja ali celo izgubljanja zahtev.

V pričujočem delu bomo obravnavali simulacije diskretnih dogodkov (angl. *discrete event simulation*), saj bodo osnovne entitete opazovanja zahteve (paketi), ki jih lahko glede na čas porajanja in njihovo naravo (velikost in tip) obravnavamo kot diskretne entitete.

Na sliki 2.1 je predstavljen ciklični proces simulacijske analize. Cikel se začne pri sistemu, ki ga opazujemo, sledijo pa postavitev modela, izvedba posameznega simulacijskega eksperimenta, analiza simulacijskih rezultatov, zaključki o opazovanem sistemu, odtod pa z eventuelnimi spremembami dopolnimo opazovani sistem v luči spremembe modela. Za vse faze cikla razen izvedbe simulacijskega eksperimenta je zadolžen človek.



Slika 2.1: Grafični prikaz cikličnega postopka simulacijske analize.

2.2 Koraki za izgradnjo cikličnega procesa simulacijske analize

Simulacijski cikel temelji na *sistemskih entitetah*, *vhodnih spremenljivkah*, *zmogljivostnih metrikah* in nenazadnje na *relacijah* med sistemskimi entitetami. Pod

sistemskimi entitetami smatramo npr. strežnike in vrste, pod vhodnimi spremenljivkami npr. intenzivnosti prihajanja zahtev in intenzivnosti strežnih procesov posameznih strežnikov, pod zmogljivostnimi metrikami npr. povprečni čakalni čas v opazovani vrsti ali maksimalna dolžina vrste, pod relacijami pa vrstni red povezovanja sistemskih entitet.

Potrebni koraki za izgradnjo cikličnega procesa simulacijske analize so sledeči:

- identifikacija opazovanega sistema ali problema (če sistem še ne obstaja);
- formulacija opazovanega sistema ali problema: potrebno je postaviti meje opazovanega sistema, definirati zmogljivostne metrike, ki bodo služile kvantitativnemu ocenjevanju različnih konfiguracij sistema, zasnovati različne konfiguracije sistema, ki se jih bo ocenjevalo in določiti ciljno publiko, ki ji bodo rezultati analize predstavljeni;
- zbiranje podatkov o opazovanem sistemu: tu so mišljeni podatki želenih sistemskih parametrov, po drugi plati pa tudi podatki (vhodne spremenljivke) o vseh entitetah, ki bodo tvorile sistem; na tem mestu je potrebno identificirati vse variabilne ali naključne faktorje v opazovanem sistemu (večina sistemov se sooča s stohastičnostjo vhodnih spremenljivk);
- formulacija in razvoj modela: potrebno je zgraditi mrežni diagram sistema, s čimer identificiramo tok zahtev po modelu;
- validacija modela: potrebno je izvesti primerjavo dinamike simulacije ob vnaprej podanih vhodih z definiranim vplivom z realnimi ali želenimi karakteristikami opazovanega sistema;
- dokumentiranje modela zaradi kasnejših nadgradenj: potrebno je natančno opisati vse zgoraj naštete korake;
- izbira simulacijskega načrta: potrebno je zasnovati serijo eksperimentov (simulacij), ki naj bi vodili do izboljšanja zmogljivosti ali odprave napak v sistemu; pozorni moramo biti na število ponovitev eksperimentov v razmerah, ko imamo opravka z variabilnostjo vhodnih spremenljivk; v tem primeru mora biti število eksperimentov večje (običajno od 3 do 5 ponovitev);
- določitev začetnih in robnih pogojev simulacij: s tem imamo v mislih npr. začetno število zahtev v sistemu, opcijsko variabilnost vhodnih spremenljivk (npr. intenzivnosti strežbe), dolžino eksperimenta, čas ogrevanja simuliranega sistema, način izbire pseudo naključnih števil, ki nam pomagajo pri izbiri variabilnih vhodnih spremenljivk, itd.;
- izvajanje simulacijskih eksperimentov;
- interpretacija simulacijskih rezultatov: na tem koraku moramo numerične izsledke pridobljene s simulacijo pretvoriti v zaključke, ki bodo v pomoč pri izdelavi sklepa o morebitnih spremembah;

- izdelava sklepa o morebitnih spremembah opazovanega sistema.

2.3 Prednosti in slabosti modeliranja in izvajanja simulacij

Prednosti, ki jih prinaša cikel modeliranja in simulacij, lahko strnemo v naslednje alineje:

- pridobivanje boljšega razumevanja opazovanega sistema,
- možnost testiranja hipotez o sistemu,
- možnost analize predvidene sistemske dinamike ali dinamike v stresnih razmerah,
- identifikacija *vodilnih* vplivnih vhodnih spremenljivk,
- identifikacija ozkih grl v sistemu,
- možnost uporabe različnih ocenjevalnih metrik.

Slabosti omenjenih postopkov so predvsem v dodatnem potrebnem času in razpoložljivosti ustreznih resursov (npr. ljudi z znanjem), kar posledično lahko vodi v višje stroške povezane z nadgradnjo ali zasnovo sistema.

2.4 Izbira simulacijskega programskega okolja

Programska modelirno simulacijska okolja delimo na dve skupini in sicer na *simulacijske jezike* in na *aplikacijsko orientirane simulatorje*. Primeri prvih so GPSS (zgodovinsko najstarejši), Arena, Simscript in pa trenutno aktualni OMNet++, primeri drugih pa Simprocess, OPNET, COMNET, itd. V večini primerov velja, da so prva okolja javno dostopna in brezplačna, druga pa uporabniško prijazna, usmerjena le na določen tip uporabe in cenovno relativno draga.

Poglavje 3

Teorija strežbe

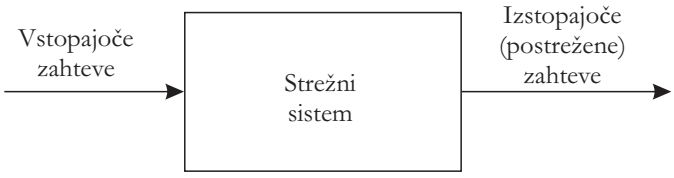
3.1 Model strežnega sistema

Teorija strežbe se je začela razvijati v začetku prejšnjega stoletja predvsem zaradi potreb obvladovanja načrtovanja zmogljivosti klasične telefonije, kasneje pa tudi zaradi potreb hitrega razvoja računalniških sistemov. Prvi obsežnejši pregled teorije strežbe je opravil Leonard Kleinrock, njegov povzetek pa najdemo v delu [2]. Temeljni gradniki omenjene teorije so *zahteve*, *čakalne vrste* in *strežniki*, katerih pomen v povezavi z računalniškimi omrežji smo spoznali že v prejšnjem poglavju. Teorija strežbe nam bo v kontekstu pričujočega dela v pomoč predvsem pri ugotavljanju zmogljivosti računalniških omrežij.

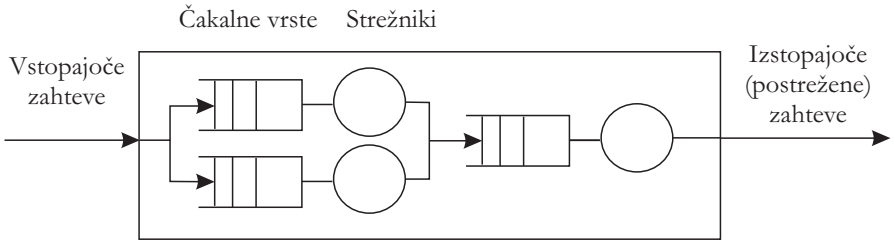
Osnovni objekt pričujočega poglavja je *model strežnega sistema*. Pod tem pojmom si predstavljamo poljuben zaključen sistem, v katerega na vhodni strani vstopajo zahteve ter od njega pričakujejo eno ali več strežb (obdelav), ki jih strežni sistem ponuja. V modelih se nahajajo strežniki, pred njimi pa čakalne vrste. Vezave poti in strežnikov s čakalnimi vrstami so poljubne. Na izhodni strani modela postrežene zahteve model sistema zapuščajo.

V vsak model strežnega sistema iz zunanjega okolja vstopajo zahteve. V domeni računalniških in komunikacijskih sistemov jih imenujemo tudi *vhodno breme* sistema [3]. Vstopajoče zahteve skušajo v okviru modela sistema priti do ustreznih storitev oziroma obdelav (*strežbe*), pri čemer morajo biti za to na razpolago ustrezni *strežniki*. Zmogljivost strežnikov običajno podajamo s kriterijem *intenzivnosti strežbe* (v domeni računalništva tudi intenzivnost procesiranja), s katero se obdelujejo vstopajoče zahteve. Podajamo jo v številu obdelanih zahtev na časovno enoto.

Ker je število strežnikov v realnih strežnih sistemih običajno omejeno, se v modelu pred posameznimi strežniki običajno nahajajo *čakalne vrste*, v katerih zahteve čakajo na prost strežnik (strežbo). Osnovna shema modela strežnega sistema je podana na sliki 3.1, na sliki 3.2 pa sta predstavljena grafična primitiva čakalne vrste in strežnika, povezana v kompleksnejši primer modela strežnega sistema z dvema možnima potema, tremi strežniki in tremi čakalnimi vrstami.



Slika 3.1: Osnovna shema modela strežnega sistema.



Slika 3.2: Primer strukture notranjosti modela strežnega sistema.

V kontekstu realnih računalniških sistemov zahteve igrajo vlogo paketov v računalniških omrežjih, zahteve po procesiranju podane s strani aplikacije proti procesorju v opazovanem računalniškem sistemu itd. Tipični primeri strežnikov so v realnih računalniških sistemih procesorji, registri, vodila, prenosni komunikacijski kanali, usmerjevalniki v računalniških omrežjih, itd.

3.2 Čakalne vrste v strežnih sistemih

Prva pomembna značilnost čakalnih vrst je njihova *velikost*, ki ponazarja njihovo končnost (omejenost na m čakalnih mest) ali neskončnost. Na hipotetično neskončne čakalne vrste pristajamo, ker se nam s tem pri matematični analizi poenostavijo modeli. Po drugi plati v realnem sistemu neskončne čakalne vrste ne moremo realizirati, zato so za nas zanimive tudi končne čakalne vrste. V primeru uporabe slednjih lahko pride do tega, da se čakalna vrsta zapolni, kar nas lahko pripelje do izgubljanja novoprispelih zahtev. V tem primeru smatramo, da je vsaka novoprispela zahteva za posamezen strežnik izgubljena.

Druga pomembna značilnost čakalnih vrst je strežna disciplina, ali vrstni red jemanja zahtev iz vrste. Osnovni tipi strežnih disciplin so naslednji:

- FIFO disciplina (angl. *first in, first out*): vrstni red jemanja zahtev iz vrste je enak vrstnemu redu prihajanja v vrsto;
- LIFO disciplina (angl. *last in, first out*): vrstni red jemanja zahtev iz vrste je obraten vrstnemu redu prihajanja v vrsto; tipičen primer tovrstne uporabe v računalništvu najdemo pri realizaciji sklada (angl. *stack*);

- prioritetna disciplina: vrstni red jemanja zahtev iz vrste je pogojen s prioritetami posameznih zahtev; iz vrste se venomer jemlje zahtevo z najvišjo prioriteto, ne glede na njeno mesto v vrsti; tipičen primer tovrstne uporabe v računalniških komunikacijah je v različnih prioritetah paketov, ki jih generirajo visokonivojski protokoli;
- naključna disciplina (angl. *random*): vrstni red jemanja zahtev iz vrste je naključen;
- SJF disciplina (angl. *shortest job first*): iz vrste se venomer vzame zahtevo, ki ima najkraši čas obdelave;
- LJF disciplina (angl. *largest job first*): iz vrste se venomer vzame zahtevo, ki ima najdaljši čas obdelave.

V okviru razdelka o čakalnih vrstah ne smemo pozabiti na metodologijo deljenja časovnih rezin (angl. *time sharing*). V tem primeru se razpoložljiva enotska rezina strežnega časa strežnika dodeljuje prvočakajoči zahtevi. Ni nujno, da bo zahteva po dodelitvi časovne rezine dokončno postrežena. Če ne bo, se vrne na konec čakalne vrste in čaka po FIFO principu na dodelitev nove časovne rezine.

3.3 Strežna enota in strežna mreža

Pod pojmom *strežne enote* si predstavljamo model strežnega sistema, ki ima sledeče lastnosti:

- strežna enota ponuja le en tip strežbe;
- v strežni enoti imamo n ($n \geq 1$) paralelno vezanih strežnikov, pri čemer so le ti funkcionalno ekvivalentni; vhodna zahteva bo deležna natanko ene strežbe v natanko enem od n strežnikov;
- pred paralelno vezavo strežnikov se običajno nahaja čakalna vrsta.

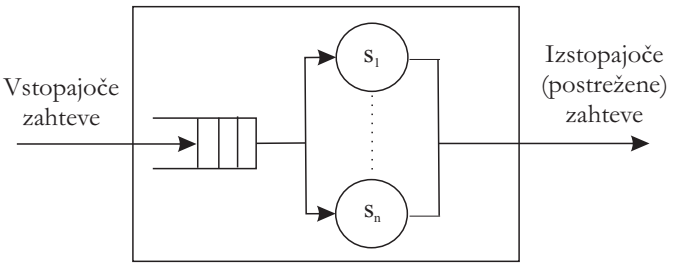
Shema opisane strežne enote se nahaja na sliki 3.3.

Pod pojmom *strežne mreže* si predstavljamo model strežnega sistema, ki je sestavljen iz poljubne zaporedne ali vzporedne vezave m strežnih enot ($m > 1$). V strežni mreži se tako lahko pojavi več kot ena čakalna vrsta in zaporedje strežb, skozi katere naj bi prišla zahteva.

V pričujočem poglavju bomo večino prostora namenili obravnavi strežnih enot, na samem koncu pa se bomo osredotočili tudi na strežne mreže.

3.4 Kendallova notacija strežnih enot

Eno od najbolj uveljavljenih klasifikacij strežnih enot je leta 1953 postavil britanski statistik D. G. Kendall [4]. Uvedel je posebno notacijo, kjer poljuben tip strežne enote opiše s petorčkom iz izraza



Slika 3.3: Shema strežne enote.

$$A \;/\; B \;/\; m \;/\; k \;/\; P. \tag{3.1}$$

Pomeni posameznih parametrov petorčka so sledeči:

- A: verjetnostna porazdelitev medprihodnih časov vstopajočih zahtev,
- B: verjetnostna porazdelitev strežnih časov zahtev,
- m: število paralelno vezanih strežnikov,
- k: kapaciteta strežne enote, ki predstavlja maksimalno število zahtev, ki jih enota lahko sprejme; izraža se kot $k = m + len(queue)$, kjer izraz $len(queue)$ predstavlja dolžino edine čakalne vrste (privzeta vrednost je neskončna kapaciteta, kar ob končnem številu strežnikov neposredno implicira hipotetično neskončno dolžino vrste) in
- P: velikost populacije, ki vstopa iz zunanjega okolja v strežno enoto (privzeta vrednost je neskončnost populacije).

Za parametra A in B uporabljamo oznake z naslednjimi pomeni:

- M: eksponentna porazdelitev markovskega (Poissonovega) procesa,
- D: deterministična porazdelitev,
- E_k: Erlangova porazdelitev reda k in
- G: splošna verjetnostna porazdelitev.

Parametre m, k in P zapisujemo s celoštevilčnimi vrednostmi. Zadnja dva parametra k in P navajamo le, če sta različna od privzetih vrednosti, navedenih v predhodnih alinejah.

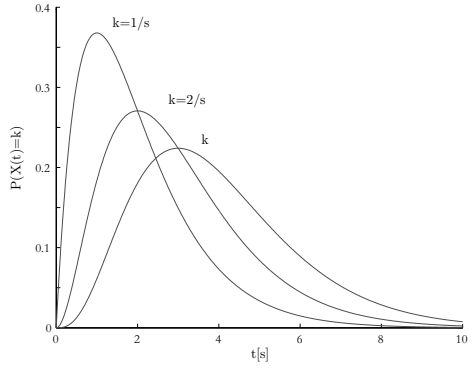
Za modeliranje medprihodnih in strežnih časov je pri računalniških omrežjih najpogosteje uporabljena Poissonova porazdelitev. S pomočjo slednje lahko modeliramo naključno porajanje dogodkov z vnaprej določeno intenzivnostjo porajanja dogodkov λ v opazovanem časovnem intervalu $(0, t)$. V nadaljevanju

dela si bomo natančneje ogledali primere strežnih enot $M/M/1$, $M/M/1/s$, $M/M/m$, $M/M/m/m$ in $M/M/m/m/c$.

Poissonovo verjetnostno porazdelitev podajamo z enačbo

$$P(X(t) = k) = \frac{(\lambda t)^k}{k!} e^{-\lambda t}, \tag{3.2}$$

kjer λ predstavlja intenzivnost porajanja zahtev v časovnem intervalu $(0, t)$. Slika 3.4 prikazuje funkcije verjetnosti (angl. *probability mass function*) ob konstantni intenzivnosti porajanja λ za različne vrednosti k.



Slika 3.4: Funkcije verjetnosti za različne vrednosti parametra k ob konstantni intenzivnosti porajanja λ.

3.5 Osnovne značilnosti strežnih enot

Kot smo povedali že predhodno, je strežna enota sestavljena iz n paralelno vezanih *strežnikov* ($n \geq 1$) in iz natanko ene *čakalne vrste* poljubne dolžine (njena dolžina je lahko tudi 0), ki se formira pred procesom strežbe. Osnovno shemo strežne enote smo predstavili na sliki 3.3. V nadaljevanju si bomo ogledali dodatne značilnosti omenjenih modelov.

3.5.1 Vhodni proces

Brez porajanja zahtev, ki predstavljajo breme modeliranega strežnega sistema, bi bil slednji ves čas nezaseden (nezaposlen) in kot tak nezanimiv za opazovanje. Osnovna značilnost porajanja zahtev je njihova *intenzivnost porajanja*. Označujemo jo z λ, merimo pa s povprečnim številom prispelih zahtev na časovno enoto. Intenzivnost porajanja definira *medprihodne čase* med posameznimi zahtevami. Ti so lahko skozi čas konstantni (vhodni proces ima značilnost deterministične porazdelitve D), ali pa upoštevajo določeno verjetnostno porazdelitev (vhodni proces ima npr. značilnost Poissonove porazdelitve, ki jo označujemo z M). V

nekaterih primerih je lahko intenzivnost časovno spremenljiva, kar je na primer lahko pogojeno z velikostjo zunanje populacije. V večini primerov predpostavljamo, da je populacija zahtev neskončna ($P = \infty$). Na ta način imamo opravka s časovno neodvisno ali statično intenzivnostjo prihajanja zahtev.

3.5.2 Strežniki

Posamezen strežnik lahko obdeluje istočasno le eno zahtevo. Osnovna značilnost strežnika je *intenzivnost obdelave zahtev*, ki jo označujemo z μ. Merimo jo s številom obdelanih zahtev na časovno enoto. Inverzna vrednost intenzivnosti procesiranja zapišemo kot $\overline{x}$ in predstavlja *povprečni strežni čas* za strežbo posamezne zahteve. Tudi strežni časi so lahko konstantni, ali pa upoštevajo določeno porazdelitev.

3.5.3 Kapaciteta strežne enote

Kapaciteta strežne enote je enaka maksimalnemu številu zahtev k, ki se lahko istočasno zadržujejo v strežni enoti. Izraža se kot vsota dolžine čakalne vrste in števila strežnikov v strežni enoti. Če je kapaciteta vrste neskončna, kar nam poenostavi izračune pri analitičnem pristopu ocenjevanja dinamike zahtev, je neskončna tudi kapaciteta strežne enote. Slednje je običajno lahko hipoteza v postopkih modeliranja, v realnem sistemu pa neskončne čakalne vrste ne moremo realizirati.

3.5.4 Izhodni proces

Osnovna značilnost mesta izstopa zahtev iz strežne enote je *intenzivnost strežbe zahtev*, ki je odvisna tako od spremenljivke λ (vhodnega procesa), kot tudi od spremenljivke μ. Če imamo v strežni enoti le en strežnik, potem se intenzivnost strežne enote ne razlikuje od intenzivnosti obdelave posameznega strežnika. V primeru, da imamo paralelno zvezanih m ekvivalentnih strežnikov ($m > 1$), je intenzivnost obdelave strežne enote m-krat večja od intenzivnosti obdelave posameznega strežnika.

3.6 Metrike za ocenjevanje strežnih enot

Za ocenjevanje zmogljivosti modelov večjih strežnih sistemov potrebujemo ustrezne podatke o zmogljivosti njegovih posameznih sestavnih delov - strežnih enot. V ta namen bomo v pričujočem razdelku vpeljali metrike za ocenjevanje strežnih enot.

3.6.1 Osnovne numerične metrike

Osnovne numerične metrike opazovane strežne enote, ki so z vidika modeliranja zanimive za analitika, so po [4] sledeče:

- $N(t)$: število zahtev v strežni enoti v opazovanem času t,
- N: povprečno število zahtev v strežni enoti v časovnem intervalu $[0, t]$,
- $N_q(t)$: število zahtev v čakalni vrsti v opazovanem času t,
- N_q: povprečno število zahtev v čakalni vrsti v časovnem intervalu $[0, t]$,
- $N_s(t)$: število zahtev v strežniku v opazovanem času t,
- N_s: povprečno število zahtev v strežniku v časovnem intervalu $[0, t]$,
- T_i: čas prebivanja i-te zahteve v strežni enoti,
- T: povprečni čas prebivanja zahteve v strežni enoti,
- W_i: čas prebivanja i-te zahteve v vrsti strežne enote,
- W: povprečni čas prebivanja zahteve v vrsti strežne enote,
- x_i: čas strežbe i-te zahteve v strežni enoti,
- $\overline{x}$: povprečni čas strežbe zahteve v strežni enoti,
- $P_k(t)$: verjetnost nahajanja k zahtev v strežni enoti v opazovanem času t,
- P_k: stacionarna verjetnost prebivanja k zahtev v strežni enoti.

Za skupno število zahtev v sistemu veljata izraza

$$N(t) = N_q(t) + N_s(t), \tag{3.3}$$

$$N = N_q + N_s, \tag{3.4}$$

pri čemer prvi glasi na opazovano časovno točko t, drugi pa na časovno povprečje. Za čas prebivanja zahteve v strežni enoti veljata izraza

$$T_i = W_i + x_i, \tag{3.5}$$

$$T = W + \overline{x}, \tag{3.6}$$

pri čemer prvi glasi na opazovano i-to zahtevo, drugi pa na časovno povprečje.

3.6.2 Littlovo pravilo

Ena od osnovnih značilnosti strežnih enot je veljavnost relacije

$$N = T * \lambda, \tag{3.7}$$

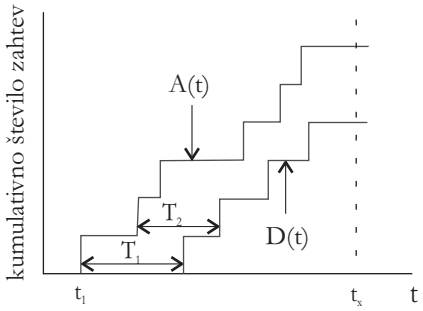
ki jo imenujemo *Littlovo pravilo* [4]. Omenjena relacija je ob pravilni interpretaciji spremenljivk N, λ in T uporabna za vse vrste strežnih enot.

Dokažimo splošno veljavnost relacije. Opazujoč število vstopajočih in izstopajočih zahtev iz strežne enote pridemo do stopničastih funkcij $A(t)$, ki predstavlja kumulativno število prispelih zahtev v časovnem intervalu $[0, t_x]$ in $D(t)$,

ki predstavlja kumulativno število zahtev, ki so v časovnem intervalu $[0, t_x]$ izstopile iz strežne enote. Za navedeni funkciji velja relacija

$$\forall t \in T : A(t) \geq D(t). \tag{3.8}$$

Če obe kumulativni funkciji ponazorimo s sliko 3.5 ugotovimo, da je v splošnem $N(t)$ določena z izrazom (3.9) kot razlika površin obeh funkcij.



Slika 3.5: Kumulativni funkciji prihodov in odhodov iz sistema.

$$\int_0^{t_x} N(t)dt = \tag{3.9a}$$

$$= \int_0^{t_x} A(t)dt - \int_0^{t_x} D(t)dt = \tag{3.9b}$$

$$= \sum_{k=1}^{D(t_x)} T_k * 1 + \sum_{k=D(t_x)+1}^{A(t_x)} (t_x - t_k) * 1, \tag{3.9c}$$

Pri tem t_k predstavlja čas rojstva posamezne zahteve, T_k pa celotni čas njenega prebivanja v strežni enoti. Odtod lahko pridemo do izraza (3.10), kjer levo stran enačbe delimo z dolžino opazovanega časovnega intervala t_x. Tako leva stran tega izraza določa povprečno vrednost števila zahtev skozi opazovani čas. Na desno stran izraza (3.10) prenesemo vsoto iz izraza (3.9), ki jo utežimo z enoto in delimo z dolžino intervala kot na levi strani izraza.

$$\frac{1}{t_x} * \int_0^{t_x} N(t)dt = \left(\sum_{k=1}^{D(t_x)} T_k + \sum_{k=D(t_x)+1}^{A(t_x)} (t_x - t_k) \right) * \frac{1}{A(t)} * \frac{A(t)}{t_x}. \tag{3.10}$$

Levi del izraza (3.10) se spremeni v N, skrajno desni člen $\frac{A(t)}{t_x}$ v intenzivnost prihajanja zahtev λ, preostanek desnega dela enačbe pa v povprečni čas bivanja zahteve v sistemu T v časovnem intervalu $[0, t_x]$. S tem je dokaz veljavnosti Littlovega pravila končan.

Littlovo pravilo iz izraza (3.7) se nanaša na strežno enoto v celoti, lahko pa ga projeciramo le na čakalno vrsto ali strežnik. Tako pridemo do relacij

$$N_q = \lambda * W, \tag{3.11}$$

$$N_s = \lambda * \overline{x}. \tag{3.12}$$

3.6.3 Faktor uporabnosti in intenzivnost strežbe

Faktor uporabnosti ρ (angl. *utilization factor*) definiramo z relacijo

$$\rho = \frac{\lambda}{\mu}. \tag{3.13}$$

Pove nam, v kolikšni meri je opazovana strežna enota zaposlena na daljši rok. Strežne enote z značilnostjo $\rho < 1$ imenujemo *stabilne*, ostale pa *nestabilne*. Če imamo v strežni enoti več strežnikov (npr. m strežnikov), jih lahko opazujemo skozi njim lastne uporabnostne faktorje ρ_i $(1 \leq i \leq m)$. V teh primerih za opazovani i-ti strežnik veljata izraza (3.14) in (3.15), pri čemer T predstavlja dolžino časovnega intervala opazovanja.

$$\rho_i = \frac{\text{Čas zasedenosti } i\text{-tega strežnika}}{\text{Celotni čas opazovanja } i\text{-tega strežnika}}, \tag{3.14}$$

$$\rho = \frac{\frac{\lambda * T}{m} * \frac{1}{\mu}}{T} = \frac{\lambda}{m * \mu}. \tag{3.15}$$

Sorodna mera faktorju uporabnosti je *intenzivnost prometa (angl. traffic intensity)* v strežni enoti. Tudi ta se izračunava kot kvocient med intenzivnostjo prihajanja zahtev in intenzivnostjo servisiranja zahtev. Z razliko od uporabnostnega faktorja ni brez enote. V primeru intenzivnosti prometa govorimo o enoti *Erlang*. Enota enega *Erlanga* ponazarja sistem, v katerem npr. pride na eno časovno enoto natanko ena zahteva, ki se servisira eno časovno enoto. Strežne enote z intenzivnostjo prometa več *Erlangov* zahtevajo vključevanje dodatnega števila strežnikov, da bi mera padla pod 1 *Erlang*. Enota z enim strežnikom in intenzivnostjo prometa 12,4 *Erlangov* bi torej zahtevala aktiviranje nadaljnjih dvanajstih strežnikov, da bi strežba potekala nemoteno.

3.6.4 Zakon o ohranitvi pretoka

Zakon o ohranitvi pretoka skozi strežno enoto predvideva, da je intenzivnost prihajanja zahtev v sistem na daljši rok opazovanja enaka intenzivnosti odhajanja zahtev iz sistema. Upoštevajoč intenzivnosti prihodov a in odhodov b iz strežne enote lahko naredimo naslednje zaključke:

- če bi veljala relacija $a > b$, potem bi imeli v sistemu vse več zahtev (število zahtev bi v čakalnih vrstah vseskozi naraščalo) in s časom bi postal nestabilen, saj poljubno dolgih čakalnih vrst v realnih sistemih ne moremo realizirati; pravimo, da tak sistem preide v *nasičenje*, saj je število zahtev v sistemu vse večje;

- če bi veljala relacija $a < b$, bi to pomenilo, da se nam v notranjosti generirajo nove zahteve, česar v svoji definiciji strežna enota ne predvideva.

Tako lahko naredimo sklep, da pri dovolj dolgem času opazovanja stabilnega strežnega sistema velja relacija $a = b$.

3.7 Poissonov proces

Za večino realnih sistemov velja, da so medprihodni in strežni časi porazdeljeni eksponentno [4]. Povedano drugače, sta tako prihajalni, kot tudi strežni proces Poissonova procesa.

Omenjeni proces si najlažje interpretiramo kot proces štetja naključno porajajočih se k dogodkov v časovnem intervalu $[0, t]$. Naključna spremenljivka $X(t)$, ki predstavlja število med seboj neodvisno porajajočih se dogodkov, je porazdeljena po Poissonovi porazdelitvi, ki je predstavljena v izrazu (3.16). Produkt λt imenujemo povprečje Poissonove naključne spremenljivke, predstavlja pa povprečno število pojavitev dogodkov v časovnem intervalu $[0, t]$.

$$P(X(t) = k) = \frac{(\lambda t)^k}{k!} e^{-\lambda t}. \tag{3.16}$$

Poissonov proces je poseben primer binomske porazdelitve Bernoullijevega eksperimenta [4]. Predpostavimo, da je časovni interval $[0, t]$ razdeljen na n podintervalov, ki so tako kratki, da se v njih lahko porodi le en dogodek. Glede na k porajanj dogodkov se binomska porazdelitev izračuna po izrazu (3.17), pri čemer je p verjetnost porajanja dogodka.

$$P\left[k \textit{ dogodkov v } n \textit{ podintervalih}\right] = \left(\begin{array}{c} n \\ k \end{array} \right) p^k (1-p)^{n-k}. \tag{3.17}$$

Če na intervalu $[0, t]$ povečamo število podintervalov n in istočasno zmanjšamo verjetnost p, tako da povprečno število porajanih dogodkov ostane nespremenjeno (to število je $np = \lambda t$), dobimo Poissonovo porazdelitev (3.18).

$$P\left[k \textit{ dogodkov v } n \textit{ podintervalih}\right] = \lim_{n\to\infty} \left(\begin{array}{c} n \\ k \end{array} \right) \left(\frac{\lambda t}{n}\right)^k \left(1 - \frac{\lambda t}{n}\right)^{n-k} =$$

$$= \frac{(\lambda t)^k}{k!} \left[\lim_{n\to\infty} \frac{n(n-1)...(n-k+1)}{n^k}\right] \left[\lim_{n\to\infty} \left(1 - \frac{\lambda t}{n}\right)^n\right] =$$

$$= \frac{(\lambda t)^k}{k!} \left[\lim_{n\to\infty} \left\{\left(1 - \frac{\lambda t}{n}\right)^{\frac{n}{\lambda t}}\right\}^{-\lambda t}\right] = \frac{(\lambda t)^k}{k!} e^{-\lambda t}. \tag{3.18}$$

3.7.1 Poissonov proces z vidika prihajanja zahtev

Porajanje dogodkov na časovni osi lahko z vidika teorije strežbe izenačimo s pojmom prihajanja ali porajanja zahtev v strežni sistem [4]. Tako lahko pogled iz prejšnjega razdelka na Poissonov proces spremenimo. Poissonov proces nam poda verjetnost za k pojavitev zahtev na opazovanem časovnem intervalu $[0, t]$, pri čemer velja, da je začetno število zahtev 0 ($P(X(0) = 0) = 1$). Za Poissonov proces morata biti poleg povedanega izpolnjena še naslednja pogoja:

- Porazdelitev števila porajanj zahtev v časovnem intervalu je odvisna le od dolžine tega intervala. Tako veljajo relacije

$$P\left[X(\Delta t) = 0\right] = 1 - \lambda\Delta t + O(\Delta t), \tag{3.19}$$

$$P\left[X(\Delta t) = 1\right] = \lambda\Delta t + O(\Delta t), \tag{3.20}$$

$$P\left[X(\Delta t) \geq 2\right] = O(\Delta t), \tag{3.21}$$

  pri čemer je $O(\Delta t)$ definiran kot funkcija z značilnostjo $\lim_{\Delta t \to 0} \frac{O(\Delta t)}{\Delta t} = 0$.

- Število porajanj zahtev v neprekrivajočih se časovnih intervalih je statistično neodvisno.

Predpostavimo, da predstavlja $P_k(t)$ verjetnost porajanja k zahtev na časovnem intervalu $[0, t]$. S pomočjo izraza (3.22) za spremembo verjetnosti po pretečenem času Δt pridemo do izrazov

$$P_k(t + \Delta t) = \sum_{i=0}^{k} P\left[(k - i)\ dogodkov\ v\ \ [0, t]\ \ in\ i\ v\ \Delta t\right], \tag{3.22}$$

$$P_0(t) = e^{-\lambda t}, \tag{3.23}$$

$$P_k(t) = \frac{(\lambda t)^k}{k!} e^{-\lambda t}. \tag{3.24}$$

3.7.2 Lastnosti Poissonovega procesa

Poissonov proces ima značilnost *superpozicije*. Ob predpostavki, da imamo k neodvisnih Poissonovih procesov in jih združimo v nov enovit proces, je tudi nov proces Poissonov proces. Intenzivnost porajanja točk novega procesa je enaka vsoti intenzivnosti porajanj posameznih Poissonovih procesov.

Eksponentna porazdelitev in Poissonov proces sta tesno povezana. Če so medprihodni časi zahtev porazdeljeni po eksponentni verjetnostni porazdelitvi, potem je število porajanih zahtev v časovnem intervalu podano s Poissonovo porazdelitvijo in proces imenujemo Poissonov proces. Obrnjeno velja, da če je število porajanih zahtev v nekem intervalu Poissonova naključna spremenljivka, potem so medprihodni časi eksponentno verjetnostno porazdeljeni in proces porajanja imenujemo Poissonov proces.

Predpostavimo, da je τ medprihodni čas. Potem velja relacija $P[\tau \leq t] = 1- P[\tau > t]$. Ker $P[\tau > t]$ predstavlja verjetnost, da do porajanja zahteve v časovnem intervalu $[0, t]$ ni prišlo (torej verjetnost $P_0(t)$), lahko naredimo sklep z izrazom

$$P[\tau \leq t] = 1 - e^{-\lambda t}. \tag{3.25}$$

Poissonov proces je *brez pomnjenja*. Če takšen proces opazujemo v neki časovni točki t, ne vemo kdaj v prihodnosti bo prispela nova zahteva, poznamo pa verjetnost porajanja te zahteve. Tudi če bi poznali časovno točko zadnjega porajanja zahteve $t - \Delta t$, časovne točke prihoda nove zahteve ne moremo določiti.

3.8 Stohastični proces

Stohastični proces je dinamični proces, ki skozi čas spreminja verjetnosti sprememb dogodkov [4]. Po definiciji je stohastični proces družina naključnih spremenljivk $\{X(t),\ t \in T\}$, ki so definirane nad verjetnostnim prostorom. Spremenljivka t predstavlja čas in izhaja iz indeksne množice T. Stohastični proces je determiniran z naslednjimi tremi parametri:

- **Prostor stanj**: Posamezno vrednost, ki jo zasede naključna spremenljivka, imenujemo *stanje procesa*, množico vseh vrednosti pa *prostor stanj procesa*. Za omenjeno *zalogo vrednosti stanj procesa* velja, da je lahko končna. V tem primeru imamo opravka s procesom *diskretnih stanj* ali *stohastično verigo*. Če zaloga stanj vsebuje končne ali neskončne intervale, potem imamo opravka s procesom *zveznih stanj*.
- **Indeksni parameter**: V kontekstu stohastičnih procesov se indeksni parameter uporablja za časovno referenciranje. Zopet imamo dve možnosti. Če sistem menja stanja v diskretnih korakih govorimo o procesu z *diskretnim časom*. Takšen proces imenujemo tudi *stohastična sekvenca*. Običajno indeksiramo tovrstne sisteme z notacijo $\{X_n\}$. Če lahko proces stanje zamenja kadarkoli na časovni osi, potem govorimo o procesu z *zveznim časom*, kar bomo označevali z notacijo $\{X(t)\}$. Menjavo stanja procesa imenujemo *tranzicija* ali *prehajanje*.
- **Statistična odvisnost**: Pojem poudarja statistično odvisnost najmanj ene od naključnih spremenljivk od ostalih članov množice spremenljivk.

V tabeli 3.1 je predstavljena klasifikacija stohastičnih procesov glede na prostor stanj in indeksni parameter (čas). Z vidika stohastičnih procesov nas večinoma zanima verjetnost $X(t)$, da bomo v času t dosegli stanje i $(P[X(t) = i])$.

3.9 Diskretni časovni Markovski proces

Markovski proces je posebna oblika stohastičnega procesa za katerega velja, da nima pomnjenja. Povedano drugače je vpliv na naslednjo spremembo ali me-

Čas/Prostor stanj	*Diskretni*	*Zvezni*
Diskretni	Diskretna časovna stohastična veriga	Diskretni časovni stohastični proces
Zvezni	Zvezna časovna stohastična veriga	Zvezni časovni stohastični proces

Tabela 3.1: Klasifikacija stohastičnih procesov glede na prostor stanj in indeksni parameter.

njavo stanja pogojen le s trenutnim stanjem procesa, ne pa s predhodnimi stanji istega procesa. Slednje pomeni, da je vrednost vsake naključne spremenljivke tega procesa odvisna zgolj od vrednosti nekih naključnih spremenljivk procesa na predhodnem koraku in robnih pogojev procesa. Diskretnost v časovnem smislu smo že razložili v razdelku o stohastičnih procesih. Ime procesa je povzeto po ruskem matematiku Andreju Andrejeviču Markovu, ki je prvi raziskoval tovrstne sisteme. Če je čas diskreten, tovrstne procese imenujemo *Markovske verige*.

Kot primer Markovskega procesa navedimo sekvenco desetih zaporednih metov kovanca. V spremenljivko x_i beležimo izide metov (1 za grb in 0 za cifro), v spremenljivko y_i pa kumulativo spremenljivke x_i ($y_0 = 0; y_{i+1} = y_i + x_{i+1}$). Spremenljivka x_i je tako stohastična, spremenljivka y_i pa označuje markovsko lastnost procesa.

3.9.1 Definicija Markovskih verig

Definicija 1 *Stohastično sekvenco* $\{X_k, k \in T\}$ *imenujemo Markovska veriga, če velja pogojna verjetnost iz izraza (3.26) za vsak* i, j *in* k.

$$P\left[X_{k+1} = j \mid X_0 = i_0, X_1 = i_1, ..., X_k = i_k\right] =$$
$$= P\left[X_{k+1} = j \mid X_k = i_k\right] = p_{ij} \tag{3.26}$$

Spremenljivka p_{ij} predstavlja verjetnost prehajanja iz i-tega v j-to stanje. V splošnem je p_{ij} lahko odvisna od koraka. Če te odvisnosti ni, govorimo o *konstantnih prehajalnih verjetnostih*, takšno verigo pa imenujemo *časovna homogena* Markovska veriga. V pričujočem delu bomo obravnavali samo časovno homogene Markovske verige. Običajno množico stanj indeksiramo z vrednostmi nenegativnih celih števil $\{0, 1, 2, ...\}$.

Še enkrat ponovimo, da je verjetnost naslednjega stanja v verigi odvisna le od trenutnega stanja in ne od tega, kako smo do njega prišli. Oznaka $X_k = j$ pomeni, da je veriga na k-tem koraku v stanju j. Verjetnost verige v stanju j bomo označevali z izrazom

$$\pi_j^{(k)} \equiv P\left[X_k = j\right]. \tag{3.27}$$

Veriga skozi diskretne časovne korake menja verjetnosti stanj. Menjave stanj imenujemo tudi prehajanja. Grafično tovrstna prehajanja predstavljamo z *di-*

agrami prehajanja stanj. Prehajanja med stanji so možna le ob vnaprej določenih korakih. Ob predpostavki, da ob začetku določimo startno točko verige (začetne verjetnostno posameznih stanj), lahko kasneje na osnovi izraza (3.28) izračunamo verjetnost nahajanja v poljubnem stanju na poljubnem časovnem koraku v prihodnosti.

$$P\left[X_{k+1}=j\right]=\sum_{i=0}^{\infty}P\left[X_k=i\right]P\left[X_{k+1}=j\mid X_k=i\right]=\sum_{i=0}^{\infty}\pi_i^{(k)}p_{ij}. \tag{3.28}$$

3.9.2 Matrični zapis verjetnosti stanj

Izraz (3.28) lahko najpregledneje zapišemo v obliki kvadratne matrike, katere dimenzije so odvisne od števila stanj Markovske verige. Imenovali jo bomo *matrika verjetnosti prehajanj* M. Sestavljajo jo posamezne verjetnosti prehodov med stanji

$$M=(p_{ij}),\ 1\leq i\leq n,\ 1\leq j\leq n. \tag{3.29}$$

Za tovrstne matrike veljata relaciji

$$\forall i:\sum p_{ij}=1,\ 1\leq j\leq n, \tag{3.30}$$

$$\forall i,j:p_{ij}\geq 0. \tag{3.31}$$

Za izračune na osnovi Markovske verige poleg matrike verjetnosti prehajanj potrebujemo še začetni vektor, ki nam poda verjetnosti posameznih stanj verige na začetku opazovanja procesa. V vsakem trenutku lahko za Markovsko verigo določimo verjetnosti posameznih stanj. To lahko ponazorimo z vrstičnim vektorjem $\pi^{(k)}=(\pi_0^{(k)},\pi_1^{(k)},...,\pi_n^{(k)})$, če nas zanima k-ti korak in imamo n možnih stanj sistema. Za lažji izračun iz prejšnjega primera bi tako računali posamezne vektorje po izrazih

$$\begin{aligned}\pi^{(1)}&=\pi^{(0)}M,\\ \pi^{(2)}&=\pi^{(1)}M,\\ &.....\\ \pi^{(k)}&=\pi^{(k-1)}M.\end{aligned} \tag{3.32}$$

S povratno substitucijo [4] posameznih členov π tako pridemo do izraza

$$\pi^{(k)}=\pi^{(0)}M^k, \tag{3.33}$$

kjer M^k predstavlja k-to potenco matrike. M^0 predstavlja enotsko matriko ($M^0=I$). Iz matričnih operacij vemo, da velja relacija $M^{k+1}=M^k*M$, kar lahko ponazorimo z izrazom

$$p_{ij}^{k+1}=\sum_{k=0}^{n}p_{ik}^{k}p_{kj}. \tag{3.34}$$

Omenjeno relacijo imenujemo Chapman-Kolmogorova enačba.

3.9.3 Stacionarna stanja v Markovskih verigah

V večini sistemov iz realnega okolja, ki jih modeliramo z Markovskimi verigami, obstajajo limitne vrednosti verjetnosti stanj, ki niso odvisne od začetne porazdelitve verjetnosti stanj. V tem primeru pravimo, da imajo sistemi *stacionarne verjetnosti stanj* ali *stacionarna stanja*. To je razvidno že iz izraza $\pi^{(k)} = \pi^{(0)} M^k$. Če obstaja limitna vrednost na k-tem koraku $\lim_{k\to\infty} M_{ij}^k = \pi_j$, potem lahko naredimo sledečo izpeljavo

$$\lim_{k\to\infty} \pi_j^{(k)} = \sum_i \pi_i^{(0)} M_{ij}^k = \pi_j \sum_i \pi_i^{(0)} = \pi_j. \tag{3.35}$$

Definicija 2 *Diskretna Markovska veriga $\{X_k\}$, ki je aperiodična, nereducibilna in časovno homogena, je tudi ergodična. Za ergodične Markovske verige vedno obstajajo limitne verjetnosti po izrazu*

$$\pi_j = \lim_{k\to\infty} \pi_j^{(k)} = \lim_{k\to\infty} P\left[X_k = j\right], j = 0, 1, ..., n, \tag{3.36}$$

ki so neodvisne od začetnega stanja verjetnostne porazdelitve.

Stacionarne porazdelitve π_j so enolično določene preko izrazov

$$\sum_j \pi_j = 1, \tag{3.37}$$

$$\pi_j = \sum_i \pi_i p_{ij}. \tag{3.38}$$

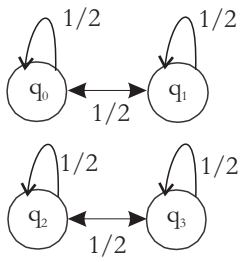
3.9.4 Značilnosti reducibilnosti in periodičnosti Markovske verige

Markovska veriga je *reducibilna*, če vsebuje več kot eno izolirano podmnožico stanj. Podmnožica stanj je izolirana, kadar iz poljubnega stanja te podmnožice ni mogoče preiti v kako drugo stanje sistema, ki ni v tej podmnožici in ko v nobeno stanje podmnožice ne vodi povezava iz ostalih stanj sistema izven podmnožice. Primer reducibilne verige je podan z matriko verjetnosti prehajanj M v izrazu 3.39, diagram prehajanja stanj pa na sliki 3.6.

$$M = \begin{bmatrix} \frac{1}{2} & \frac{1}{2} & 0 & 0 \\ \frac{1}{2} & \frac{1}{2} & 0 & 0 \\ 0 & 0 & \frac{1}{2} & \frac{1}{2} \\ 0 & 0 & \frac{1}{2} & \frac{1}{2} \end{bmatrix}. \tag{3.39}$$

Nereducibilna Markovska veriga vsebuje le eno izolirano množico stanj, v kateri so vsa stanja sistema.

Markovska veriga je *periodična* s periodo τ, če se po $n\tau$ ($n = 1, 2, ...$) korakih vrača v isto stanje sistema. V nereducibilni Markovski verigi so vsa stanja aperiodična ali periodična z isto periodo.



Slika 3.6: Diagram prehajanja stanj na osnovi podane matrike verjetnosti prehajanj M (zgled reducibilnosti).

3.10 Časovni zvezni Markovski proces

Za Markovski proces lahko vpeljemo zvezno obravnavo časa. To pomeni, da se prehajanje iz stanja v stanje lahko izvrši ob poljubnem času zveznega časovnega intervala. Še vedno pa velja, da je tovrstni proces stohastični proces $\{X(t)\}$, v katerem je prehod v novo stanje odvisen le od trenutnega stanja in ne od stanj v preteklosti. Slednje lahko zapišemo z izrazom

$$\begin{aligned} &P\left[X(t_{k+1}) = j \mid X(t_1 = i_1), X(t_2 = i_2), ..., X(t_k = i_k)\right] = \\ &= P\left[X(t_{k+1}) = j \mid X(t_k = i_k)\right],\ t_1 < t_2 < ... < t_k < t_{k+1}. \end{aligned} \tag{3.40}$$

Predpostavimo, da se Markovski proces nahaja v stanju i. Verjetnost, da proces izvede prehod v stanje j v infinitezimalnem času Δt, ne glede na to koliko časa je bil proces v stanju i, je

$$p_{ij}(t, t + \Delta t) = q_{ij}\Delta t, \tag{3.41}$$

kjer q_{ij} predstavlja intenzivnost prehajanja iz i v j. Celotna intenzivnost zapuščanja stanja i se izraža kot $\sum_{j \neq i} q_{ij}$. Čas, v katerem se sistem nahaja v stanju i, imenujemo *čas zadrževanja procesa* v stanju i.

Naj bo τ_i čas prebivanja procesa v stanju i. Predpostavimo, da časovni interval $[0, t]$ razdelimo na k enako dolgih podintervalov dolžine Δt ($t = k * \Delta t$). Če je t manjši od τ_i, potem v opazovanem intervalu ni prišlo do spremembe stanja. Verjetnost, da do menjave stanja ne pride v posameznem podintervalu, je $1 - \sum_{j \neq i} q_{ij}\Delta t$. Odtod sledi izpeljava po izrazu (3.42), kjer je $q_i = \sum_{j \neq i} q_{ij}$.

$$P\left[\tau_i > t\right] = \lim_{k \to \infty} \left[1 - \sum_{j \neq i} q_{ij}\Delta t\right]^k = \lim_{k \to \infty} \left[1 - \sum_{j \neq i} q_{ij}\frac{t}{k}\right]^k = e^{-q_i t}. \tag{3.42}$$

Ugotovimo lahko, da je porazdelitev časov zadrževanja v posameznih stanjih med prehajanji podana z eksponentno porazdelitvijo po izrazu

$$P\left[\tau_i \leq t\right] = 1 - e^{-q_i t}. \tag{3.43}$$

3.10.1 Porazdelitev verjetnosti stanj

Verjetnost zadrževanja sistema v času t v stanju j zapišemo z izrazom

$$\pi_j(t) = P\left[X(t) = j\right]. \tag{3.44}$$

Sprememba verjetnosti v infinitezimalnem času Δt je potem enaka

$$\pi_j(t+\Delta t) = \sum_{i\neq j} \pi_i(t) q_{ij} \Delta t + \pi_j(t) \left[1 - \sum_{k\neq j} q_{jk}\Delta t\right]. \tag{3.45}$$

Prvi del desnega dela izraza (3.45) ponazarja verjetnost stanja i v času t in prehod v stanje j v časovnem intervalu Δt. Drugi del desnega dela izraza (3.45) ponazarja verjetnost stanja j v času t, pri čemer do prehajanja v časovnem intervalu Δt ne pride. Če predpostavimo, da je $q_{jj} = \sum_{k\neq j} q_{jk}$ in izraz delimo z Δt, pridemo do relacije

$$\frac{d}{dt}\pi_j(t) = \sum_{i\neq j} \pi_i q_{ij} - \pi_j \sum_{k\neq j} q_{jk}, \tag{3.46}$$

preko nje pa do izrazov

$$\frac{d}{dt}\pi_j(t) = \sum_{i} \pi_i(t) q_{ij}, \tag{3.47}$$

$$\frac{d}{dt}\widetilde{\pi}(t) = \widetilde{\pi}(t)\widetilde{Q}. \tag{3.48}$$

Pri tem je $\widetilde{\pi}(t) = (\pi_1(t), \pi_2(t), ...)$, $\frac{d}{dt}\widetilde{\pi}(t) = (\frac{d}{dt}\pi_1(t), \frac{d}{dt}\pi_2(t), ...)$ in $\widetilde{Q}$ matrika intenzivnosti prehajanj ali infinitezimalni generator ($\widetilde{Q} = (q_{ij})$).

Za nereducirljiv in homogen Markovski proces z zveznim časom velja, da zanj obstaja limitna vrednost in je neodvisen od začetnega stanja verige.

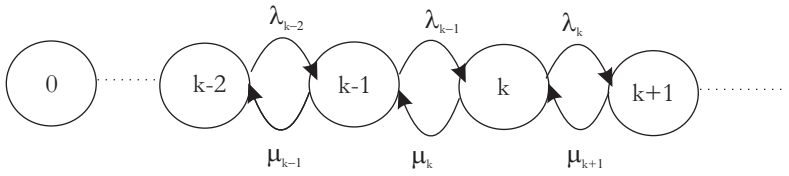
3.10.2 Matriki intenzivnosti prehajanja in verjetnosti prehajanja

Predhodno smo spoznali matriko verjetnosti prehajanja stanj M in v pričujočem razdelku matriko intenzivnosti prehajanj $\widetilde{Q}$. Obe matriki popolnoma določata Markovsko verigo in sicer prva verigo z diskretnim časom, druga pa verigo z zveznim časom. Prva matrika torej vsebuje verjetnosti, druga pa intenzivnosti. Da bi tudi v drugem objektu dobili verjetnosti, moramo posamezne intenzivnosti pomnožiti s časovnim intervalom Δt ($\Delta t * q_{ij}, \Delta t \to 0$).

3.11 Rojstno smrtni proces

Rojstno smrtni proces je posebna oblika Markovske verige, v kateri so možni prehodi iz opazovanega stanja le v sosednja stanja. Termin rojstno smrtni izvira iz analize števila zahtev v sistemu v času t. Predpostavimo, da imamo v času

t opravka s populacijo k zahtev v sistemu. Tovrstno stanje sistema v času t bomo označevali k. Prehod iz stanja k v stanje $k+1$ predstavlja rojstvo (v sistem je vstopila nova zahteva), prehod v stanje $k-1$ pa smrt (zahteva je bila uspešno servisirana in je zapustila sistem). Predpostavljamo, da se rojstvo in smrt ne moreta zgoditi istočasno, istočasno pa se ne more zgoditi istočasno več smrti ali več rojstev. Stanja procesa označujemo s celimi pozitivnimi števili $\{0, 1, 2, ...\}$, ki označujejo število zahtev v sistemu. Diagram prehajanja stanj v rojstno smrtnem sistemu je predstavljen na sliki 3.7.



Slika 3.7: Diagram prehajanja stanj rojstno smrtnega procesa.

Predpostavljajmo, da imamo v sistemu populacijo velikosti k, pri čemer λ_k predstavlja intenzivnost porojevanja zahtev v populaciji, μ_k pa intenzivnost umiranja. Velja izraz

$$\lambda_k = q_{k,k+1},\ \mu_k = q_{k,k-1}. \tag{3.49}$$

Z vpeljavo novih izrazov lahko matriko intenzivnosti prehajajanj zapišemo kot

$$Q = \begin{bmatrix} -\lambda_0 & \lambda_0 & 0 & 0 & ... \\ \mu_1 & -(\lambda_1+\mu_1) & \lambda_1 & 0 & ... \\ 0 & \mu_2 & -(\lambda_2+\mu_2) & \lambda_2 & ... \\ 0 & 0 & \mu_3 & -(\lambda_3+\mu_3) & ... \\ ... & ... & ... & ... & ... \end{bmatrix}. \tag{3.50}$$

Tako veljata relaciji

$$\frac{d}{dt}P_k(t) = -\left(\lambda_k+\mu_k\right)P_k(t) + \lambda_{k-1}P_{k-1}(t) + \mu_{k+1}P_{k+1}(t),\ k \geq 1, \tag{3.51}$$

$$\frac{d}{dt}P_0(t) = -\lambda_0 P_0(t) + \mu_1 P_1(t). \tag{3.52}$$

Poseben primer rojstno smrtnega procesa je *čisti rojstni proces*, kjer ni strežbe in posredno ne prihaja do umiranja zahtev. Zanj velja izraz

$$\forall k : \lambda_k = \lambda > 0,\ \mu_k = 0, \tag{3.53}$$

tako da se izraza (3.51) in (3.52) poenostavita v izraza

$$\frac{d}{dt}P_k(t) = -\lambda_k P_k(t) + \lambda P_{k-1}(t),\ k \geq 1, \tag{3.54}$$

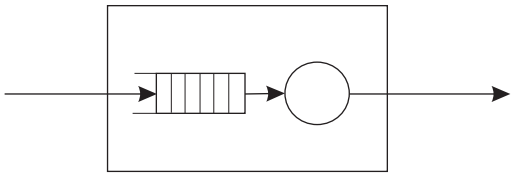
$$\frac{d}{dt}P_0(t) = -\lambda P_0(t), k = 0. \tag{3.55}$$

3.12 Markovski strežni sistemi z eno čakalno vrsto

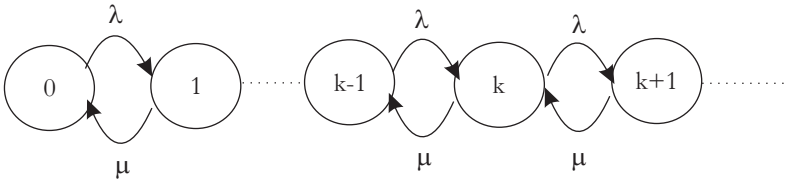
Markovski strežni sistem je strežni sistem s Poissonovim verjetnostno porazdelitvijo prihajanja zadev in eksponentno verjetnostno porazdelitvijo strežnih časov. Če imamo opravka s tovrstnim strežnim sistemom, nas običajno zanimajo vrednosti numeričnih postavk, kot so P_k, N itd.

3.12.1 Strežni sistem $M/M/1$

Najosnovnejši model sistemov iz realnega okolja je zasnovan s Poissonovim prihajalnim procesom in strežbo s strani enega strežnika z eksponentno verjetnostno porazdelitvijo strežnih časov. Sistem je predstavljen na sliki 3.8, diagram prehajanja stanj pa nas sliki 3.9.



Slika 3.8: Shema $M/M/1$ sistema.



Slika 3.9: Diagram prehajanja stanj $M/M/1$ sistema.

Za omenjeni sistem velja, da sta λ in μ neodvisni od števila zahtev v sistemu in se skozi čas ne spreminjata. Slednje pomeni, da sta neodvisni od stanja, v katerem se sistem nahaja. Za manjkajoči postavki k in P iz Kendallove notacije povzamemo privzete vrednosti, tako da ima sistem neskončno kapaciteto (neskončno dolžino vrste) in neskončno zunanjo populacijo zahtev. Neodvisnost intenzivnosti porajanja in strežbe od števila zahtev v sistemu ponazorimo z izrazom

$$\lambda_k = \lambda,\ \mu_k = \mu. \tag{3.56}$$

Iz izrazov (3.51) in (3.52) sledita posplošitvi v izrazih

$$(\lambda + \mu)P_k = \lambda P_{k-1} + \mu P_{k+1}, k \geq 1, \tag{3.57}$$

$$\mu P_1 = \lambda P_0,\ k = 0. \tag{3.58}$$

Iz tega sledita izraza

$$P_0 = 1 - \frac{\lambda}{\mu} = 1 - \rho, \tag{3.59}$$

$$P_k = (1-\rho)\rho^k. \tag{3.60}$$

Izraza (3.59) in (3.60) bi lahko izpeljali tudi na osnovi enakosti

$$P_k = \frac{\lambda}{\mu} P_{k-1} = (\frac{\lambda}{\mu})^2 P_{k-2} = ... = \rho^k P_0. \tag{3.61}$$

Osnovne značilnosti $M/M/1$ sistema so sledeče:

- Verjetnost, da imamo v sistemu n ali več zahtev, je izpeljana z izrazom

$$P\left[N \geq n\right] = \sum_{k=n}^{\infty} P_k = (1-\rho) \sum_{k=n}^{\infty} \rho^k =$$

$$= (1-\rho) \left[\sum_{k=0}^{\infty} \rho^k - \sum_{k=0}^{n-1} \rho^k \right] = (1-\rho) \left[\frac{1}{1-\rho} - \frac{1-\rho^n}{1-\rho} \right] = \rho^n. \tag{3.62}$$

- Povprečno število zahtev v sistemu je pogojeno z verjetnostmi nahajanj v posameznih stanjih po izrazu

$$N = \sum_{k=0}^{\infty} k P_k = \sum_{k=0}^{\infty} k(1-\rho)\rho^k = (1-\rho)\rho \sum_{k=0}^{\infty} k\rho^{k-1} = \frac{\rho}{1-\rho} = \frac{\lambda}{\mu - \lambda}.$$

- Povprečni čas prebivanja zahteve v sistemu T se izračuna po izrazu

$$T = \frac{N}{\lambda} = \frac{\rho}{\lambda(1-\rho)} = \frac{1}{\mu - \lambda}. \tag{3.63}$$

- Povprečno število zahtev v strežniku N_s, kar izraža zasedenost strežnika (faktor uporabnosti) je definirano z izrazom

$$N_s = \frac{\lambda}{\mu} = \rho = 1 - P_0. \tag{3.64}$$

- Povprečni čas zadrževanja zahteve v čakalni vrsti W (čakalni čas) je podan z izrazom

$$W = T - \frac{1}{\mu} = \frac{\rho}{\mu - \lambda}, \tag{3.65}$$

  ali z razliko med časom zadrževanja zahteve v sistemu in časom strežbe.

- Povprečno število zahtev v čakalni vrsti N_q se izračuna po izrazu

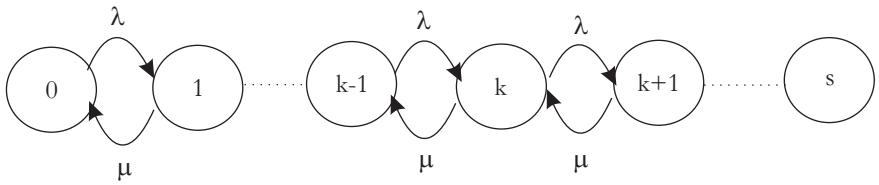
$$N_q = \lambda W = \frac{\rho^2}{1-\rho}. \tag{3.66}$$

Poseben primer $M/M/1$ sistema je *omahljiv strežni sistem*, pri katerem se intenzivnost prihajanja zahtev spreminja od odvisnosti števila zahtev v sistemu, intenzivnost procesiranja pa se ne spreminja. V splošnem omahljivi strežni sistem opišemo z izrazom

$$\lambda_k = \lambda/(k+1),\ \mu_k = \mu,\ k = 0, 1, ... \tag{3.67}$$

3.12.2 Strežni sistem $M/M/1/s$ s končno čakalno vrsto

Za sisteme iz realnega okolja, kjer ne moremo zagotoviti neskončne kapacitete, je v primerjavi z $M/M/1$ primernejši $M/M/1/s$ model, ki predvideva končno kapaciteto čakalne vrste. V tovrstnem modelu se lahko nahaja le s zahtev, pri čemer je $s-1$ zahtev v čakalni vrsti in ena v procesu strežbe. Shema omenjenega sistema je predstavljena na sliki 3.10, iz katere je razvidna omejenost sistema na desni.



Slika 3.10: Shema sistema $M/M/1/s$.

V primerjavi z $M/M/1$ sistemom, se nam osnovni numerični postavki P_0 in P_k spremenita v skladu z izrazoma

$$P_0 = \frac{1-\rho}{1-\rho^{s+1}}, \tag{3.68}$$

$$P_k = \frac{(1-\rho)\rho^k}{1-\rho^{s+1}}. \tag{3.69}$$

Sistem ima končno kapaciteto, kar pomeni, da gre za izgubni sistem.

Vstopna intenzivnost se izraža kot $\lambda' = \lambda * (1 - p_b)$, kjer p_b predstavlja verjetnost zavrnitve posamezne zahteve, ki je enaka verjetnosti stanja s. Iz veljavnosti zakona o ohranitvi pretoka lahko sklepamo, da mora biti izhodna intenzivnost γ enaka vhodni intenzivnosti. Odtod velja izraz

$$\gamma = \lambda(1 - P_b). \tag{3.70}$$

Intenzivnost zapuščanja sistema (izhodna intenzivnost) se za podani sistem lahko po drugi plati izračuna po izrazu

$$\gamma = \sum_{k=1}^{s} \mu P_k = \mu(1 - P_0). \tag{3.71}$$

Če izenačimo oba predhodna izraza tako dobimo nov izraz

$$\lambda(1 - P_b) = \mu(1 - P_0). \tag{3.72}$$

Navedeni izrazi veljajo samo za pare (μ, λ), ki so neodvisni od stanja sistema k. Glavne značilnosti enote $M/M/1/s$ so:

- Sistem je *zasičen*, ko je v njem s zahtev (maksimalno število zahtev). V tem primeru velja izraz

$$P_s = P_b = \frac{(1-\rho)\rho^s}{1-\rho^{s+1}}, \tag{3.73}$$

  ali izraz

$$P_b = \frac{1}{s+1}, \tag{3.74}$$

  če je $\rho = 1$.

- Povprečno število zahtev v sistemu N je pogojeno z verjetnostjo nahajanja v posameznem stanju in se izraža po izrazu

$$N = \sum_{k=0}^{s} kP_k = ... = \frac{\rho}{1-\rho} - \frac{\rho}{1-\rho}(s+1)P_b, \tag{3.75}$$

  če pa je $\rho = 1$, pa kot $N{=}\frac{s}{2}$.

- Povprečno število zahtev v fazi strežbe se izraža kot

$$N_s = \rho(1 - P_b). \tag{3.76}$$

- Povprečno število zahtev v čakalni vrsti se izraža kot

$$N_q = N - N_s = \frac{\rho^2}{1-\rho} - \rho\frac{\rho+s}{1-\rho}P_b. \tag{3.77}$$

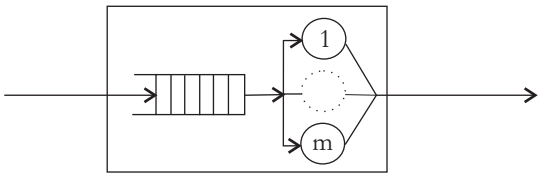
- Glede na izraz $\lambda' = \lambda * (1 - p_b)$ se povprečni čas bivanja zahteve v sistemu (T) in v čakalni vrsti (W) izračunavata kot

$$T = \frac{N}{\lambda'} = \frac{1}{\mu-\lambda} - \frac{s\rho^{s+1}}{\lambda - \mu\rho^{s+1}}, \tag{3.78}$$

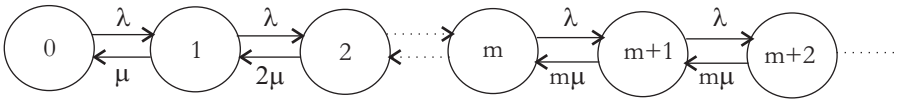
$$W = \frac{N_q}{\lambda'} = \frac{\rho}{\mu-\lambda} - \frac{s\rho^{s+1}}{\lambda - \mu\rho^{s+1}}. \tag{3.79}$$

3.12.3 Večstrežniški sistem $M/M/m$

Sistem $M/M/m$ se od $M/M/1$ razlikuje po tem, da vsebuje m paralelno vezanih funkcionalno ekvivalentnih strežnikov ($m > 1$). Vsaka zahteva je v takem sistemu obdelana natanko enkrat na poljubnem strežniku. Osnovna shema modela je prikazana na sliki 3.11, na sliki 3.12 pa shema rojstno smrtnega sistema. Tipično za tak sistem je, da mu s povečevanjem števila strežnikov m lahko povečujemo intenzivnost strežbe. Če je μ intenzivnost strežbe posameznega strežnika, imamo tako pri vsaj m zahtevah v sistemu opravka z $m\mu$ strežno intenzivnostjo, v primeru pa da je teh zahtev manj (k), pa z intenzivnostjo strežbe $k\mu$.



Slika 3.11: Shema M/M/m sistema.



Slika 3.12: Shema rojstno smrtnega procesa $M/M/m$ sistema.

Glede na sliko 3.12 lahko postavimo sistem ravnotežni enačb tipa

$$k \leq m : k\mu P_k = \lambda P_{k-1}, \tag{3.80}$$

$$k \geq m : m\mu P_k = \lambda P_{k-1}. \tag{3.81}$$

Ob vpeljavi nove spremenljivke a, ki nam omogoči preglednejši matematični zapis, za tovrsten sistem veljajo izrazi

$$\rho = \frac{\lambda}{m\mu}, \tag{3.82}$$

$$a = \frac{\lambda}{\mu}, \tag{3.83}$$

$$P_0 = \left[\sum_{k=0}^{m-1} \frac{a^k}{k!} + \frac{a^m}{m!(1-\rho)} \right]^{-1}. \tag{3.84}$$

Na osnovi vpeljanih izrazov lahko izpeljemo nov sistem ravnotežnih enačb

$$k \leq m: \ P_k = \frac{a^k}{k!} P_0, \tag{3.85}$$

$$k \geq m: P_k = \frac{a^k}{m! m^{(k-m)}} P_0. \tag{3.86}$$

Značilnosti sistema $M/M/m$ so:

- Verjetnost, da prihajajoča zahteva naleti na vse zasedene strežnike in bo morala čakati v vrsti, je pogojena z izrazom

$$P_d = \frac{P_0 a^m}{m!(1-\rho)}. \tag{3.87}$$

- Povprečno število zahtev v čakalni v vrsti se izraža kot

$$N_q = \frac{\rho}{1-\rho} P_d. \tag{3.88}$$

- Porabljen čas v čakalni vrsti se izraža kot

$$W = \frac{N_q}{\lambda}. \tag{3.89}$$

- Porabljen čas v celotnem strežnem sistemu se izraža kot

$$T = W + \frac{1}{\mu}. \tag{3.90}$$

- Celotno število zahtev v strežnem sistemu se izraža kot

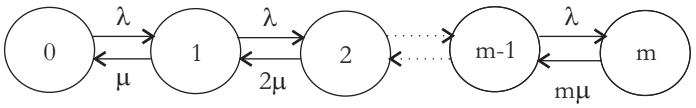
$$N = \lambda T. \tag{3.91}$$

3.12.4 Izgubni sistem $M/M/m/m$

Kot smo poudarili že v prejšnjih razdelkih, je neskončna čakalna vrsta v realnih sistemih z realizacijskega vidika neizvedljiva. Predpostavimo pa drug ekstremen primer, ko čakalne vrste sploh nimamo. Ob notaciji $M/M/m/m$ to pomeni, da imamo ob m strežnikih sistemsko kapaciteto m mest (kapaciteta sistema je torej enaka številu strežnikov, dolžina vrste pa 0). Diagram prehajanja stanj je tako povzet po $M/M/m$ sistemu, le da je na desni strani omejen (glej sliko 3.13), ker ni čakalne vrste. Iz ravnotežnih enačb

$$\lambda P_{k-1} = k\mu P_k, \tag{3.92}$$

$$P_k = P_0 \frac{a^k}{k!}, \tag{3.93}$$



Slika 3.13: Diagram prehajanja stanja v M/M/m/m sistemu.

lahko izpeljemo enačbi

$$P_0 = \left[\sum_{k=0}^{m} \frac{a^k}{k!}\right]^{-1}, P_k = \frac{a^k/k!}{\left[\sum_{k=0}^{m} \frac{a^k}{k!}\right]}. \tag{3.94}$$

Na osnovi izrazov 3.94 izpeljemo *izgubno enačbo*

$$P_b = \frac{a^m/m!}{\sum_{k=0}^{m} \frac{a^k}{k!}}, \tag{3.95}$$

pri čemer veljata izraza

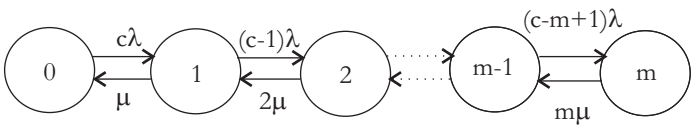
$$N = \sum_{k=0}^{m} kP_k = ... = a(1 - P_b), \tag{3.96}$$

$$T = \frac{1}{\mu}, N_q = 0, W = 0. \tag{3.97}$$

3.12.5 Engsetov izgubni sistem $M/M/m/m/c$

Engsetov izgubni sistem se od predhodno obravnavanega sistema $M/M/m/m$ razlikuje samo v velikosti vhodne populacije. Zanjo predpostavimo, da ni več neskončna ali izredno velika, temveč končna in dovolj majhna, da na vhodni strani intenzivnost prihajanja glede na število zahtev v sistemu niha. Predpostavimo, da je velikost populacije c zahtev. Če je le ta manjša od m, imamo opravka z obvladljivim sistemom, v katerem imamo ves čas vsaj en prost strežnik, kar pomeni, da je sistem z vidika zmogljivosti predimenzioniran.

V splošnem lahko stanja omenjenega sistema zopet modeliramo z rojstno smrtnim procesom, ki je predstavljen na sliki 3.14.



Slika 3.14: Diagram prehajanja stanja v $M/M/m/m/c$ sistemu.

Za tovrstni sistem lahko na osnovi ravnotežnih enačb

$$0 \leq k \leq c-1 : \lambda_k = \lambda(c-k), \tag{3.98}$$

$$0 \leq k \leq m : \mu_k = k\mu, \tag{3.99}$$

izpeljemo verjetnosti stanj

$$P_k = \binom{c}{k} a^k P_0, \tag{3.100}$$

$$P_0 = \frac{1}{\sum_{k=0}^{m} \binom{c}{k} a^k}, \tag{3.101}$$

$$P_b = P_m = \frac{\binom{c}{m} a^m}{\sum_{k=0}^{m} \binom{c}{k} a^k}, \tag{3.102}$$

in povprečno intenzivnost porajanja zahtev na vhodni strani

$$\overline{\lambda} = \sum_{k=0}^{m-1} \lambda(c-k) \left[P_0 \binom{c}{k} a^k \right] = ... = \lambda c(1-P_b) - \lambda(N - mP_b). \tag{3.103}$$

Povprečno število zahtev v sistemu se tako izraža kot

$$N = \sum_{k=0}^{m} k P_k = \overline{\lambda} T = \frac{\overline{\lambda}}{\mu}. \tag{3.104}$$

3.12.6 Povzetek sistemov strežbe tipa $M/M/\_/-/-$

Doslej smo predpostavljali, da se narava vhodnega procesa izraža po Poissonovi verjetnostni, narava strežnega procesa (časa strežbe) pa po eksponentni verjetnostni porazdelitvi. Slednja predpostavka izhaja iz sveta klasičnih telekomunikacij ali še natančneje iz narave zahtev, ki se porajajo v klicnih omrežjih. Mednje sodijo telefonski klici, pošiljanje faksov itd. Z razliko od klasičnih klicnih omrežij imamo v sedanjosti večinoma opraviti z digitaliziranim prometom, pri katerem poteka paketni prenos podatkov. S tega vidika govorimo o *podatkovnih* ali *paketnih omrežjih*. Za slednje velja, da se proces strežbe ne deklarira po eksponentni porazdelitvi strežnih časov (dolžinah paketov), temveč po drugačnih porazdelitvah. Ena od ekstremnih predpostavk je, da je dolžina paketov kar konstantne dolžine, kar vodi v *deterministične strežne čase*, še boljša predpostavka pa je, da predpostavimo na mestu strežbe *splošno porazdelitev strežnih časov*. Porazdelitev dolžine paketov, ki neposredno vpliva na porazdelitev strežnih časov, je odvisna od narave komunikacijskega protokola, ki nadzoruje promet v paketnem omrežju.

Predvsem eksponentna verjetnostna porazdelitev strežbe, ki smo jo predpostavljali v prejšnjih zgledih sistemov, je pesimistična in na osnovi drugačnih modelov lahko pridemo do boljših simulacijskih rezultatov. V primeru, da ostajamo pri njej, lahko realno upamo, da rezultati dinamike v realnem sistemu ne bodo slabši od simulacijskih rezultatov. Do sedaj predstavljeni sistemi $M/M/1$, $M/M/m$, $M/M/1/s$, $M/M/m/m$, $M/M/m/m/c$ so vključevali eno čakalno vrsto in eno strežbo in so tako sodili v kategorijo *strežnih enot*.

3.12.7 Računski zgledi s področja strežnih enot

Zgled 1 *Vzpostavili bomo novo telefonsko klicno centralo z m linijami proti globalnemu omrežju. Centrala naj bi bila sposobna vzdrževati 300 naročniških povezav. Glede na predhodno opravljeno bremensko analizo lahko pričakujemo, da bo vsak naročnik naredil 20 minut prometa dnevno. V sistemu si želimo verjetnost izgubljanja klicev, ki bo manjša ali enaka 0,02% ($P_b \leq 0,02\%$). Kolikšen naj bo m, če je maksimalna obremenitev v posamezni uri dneva 14% dnevnega prometa (angl. peak hour) in pribitek na nezanesljivost linij 10%?*

Rešitev: Predpostavimo, da klici ne morejo čakati v vrsti. Tako gre za sistem tipa M/M/m/m. Izračunamo spremenljivko $Total\_Traffic\_Load = 300$ *naročnikov* $*$ 20 *min* $*$ 0,14 / 60 *min* $= 14$ *Erlangov, ki jo dodatno povečamo za 10% (*14 *Erlangov* $*1,1 = 15,4$ *Erlanga).*

$$P_b = \frac{a^m/m!}{\sum_{k=0}^{m} \frac{a^k}{k!}} = \frac{(15,4)^m}{\sum_{k=0}^{m} \frac{a^k}{k!}}. \tag{3.105}$$

Izračun pokaže, da verjetnost P_b pade pod željeno pri m=23 linijah ($m = 23 \rightarrow P_b = 0,0167, m = 22 \rightarrow P_b = 0,0254$).

Zgled 2 *Satelitski sistem mobilne telefonije ponuja pasovno širino 1200 bps za klicne linije in 2400 bps za sporočilne (podatkovne) linije. Klici in sporočila se porajajo po Poissonovem procesu, prvi z intenzivnostjo 200 klicev na sekundo in drugi 40 prenosov na sekundo. Oboji so porazdeljeni eksponentno in sicer prvi z dolžino 54 bitov in drugi z dolžino 240 bitov. Ob zasedenosti prenosnih kapacitet se klici zavračajo, sporočila pa odhajajo v vmesnik hipotetične neskončne dolžine. Izračunaj število potrebnih klicnih kanalov (n_v), da bo verjetnost izgube klica manša od 0,02% in število potrebnih podatkovnih kanalov (n_d), da bo zamik (angl. message delay) dostave manjši od 0,115 sekunde.*

Rešitev: Klicni sistem lahko obravnavamo kot M/M/m/m sistem, podatkovni sistem pa kot M/M/m sistem.

$$\lambda_v = 200, \mu_v^{-1} = 54/1200 = 9/200, a = \frac{\lambda_v}{\mu_v} = 9, \tag{3.106}$$

$$P_b = \frac{a^m/m!}{\sum_{k=0}^{m} \frac{a^k}{k!}} = \frac{(9)^{n_v}}{\sum_{k=0}^{m} \frac{9^k}{k!}} \leq 0,02 \rightarrow n_v = 15, \tag{3.107}$$

$$\lambda_d = 40, \mu_d^{-1} = \frac{240}{2400} = 0,1, a = \frac{\lambda_d}{\mu_d} = 4 \tag{3.108}$$

$$T = \frac{1}{\mu_d} + \frac{P_d}{m\mu_d - \lambda_d} = 0,1 + \frac{P_d}{10n_d - \lambda_d} \leq 0,15 \rightarrow n_d \geq 6. \tag{3.109}$$

Zgled 3 *Imamo deset odjemalcev strežniškega sistema, ki lahko istočasno nudi tri aktivne priključke (tri aktivne seje). Vsak odjemalec poraja 6 zahtev na uro po Poissonovem procesu, vsaka zahteva pa se streže v povprečju 5 minut. Kolikšna je verjetnost izgubljanja zahtev in kolikšno je povprečno število zasedenih priključkov?*

Rešitev: Glede na povedano lahko predpostavimo, da je sistem tipa M/M/m/m/c.

$$\lambda = 6/60 = 0,1, \mu^{-1} = 5, a = 0,5, \tag{3.110}$$

$$P_b = P_m = \frac{\binom{c}{m} a^m}{\sum_{k=0}^{m} \binom{c}{k} a^k} = \frac{\binom{10}{3}(0,5)^3}{\sum_{k=0}^{3} \binom{10}{k}(0,5)^k} = 0,4651, \tag{3.111}$$

$$N = \frac{\rho}{1+\rho} c - \frac{\rho}{1+\rho}(c-m) * P_b = 2,248. \tag{3.112}$$

3.13 M/G/1 sistem

Pri tem sistemu je proces porajanja zahtev brez pomnjenja (semi Markovski model). Matematični izrazi, ki veljajo zanj so

$$N_q = \lambda * W = \frac{\lambda^2 * x^2}{2(1-\rho)} = \frac{\rho^2}{2(1-\rho)}(1 + c_b^2), \tag{3.113}$$

$$T = x + \frac{\lambda * x^2}{2(1-\rho)} = x + \frac{\rho x}{2(1-\rho)}(1 + c_b^2), \tag{3.114}$$

$$N = \rho + \frac{\lambda^2 * x^2}{2(1-\rho)} = \rho + \frac{\rho^2}{2(1-\rho)}(1 + c_b^2), \tag{3.115}$$

$$W = T - x = \rho x \frac{1 + c_b^2}{2(1-\rho)} \tag{3.116}$$

kjer c_b^2 predstavlja razmerje med standardno deviacijo in kvadratom povprečja strežnega časa. Pri $M/M/1$ tako dobimo $c_b^2 = 1$ in pri $M/D/1$ $c_b^2 = 0$.

Zgled 4 *Paketi dolžine L prihajajo v vozlišče po Poissonovi porazdelitvi z intenzivnostjo λ in čakajo na kanal s prepustnostjo D bps. Izračunaj T in N za eksponentno porazdeljene L in konstantne L. V prvem primeru predpostavimo sistem M/M/1 (a), v drugem pa M/D/1 (b).*

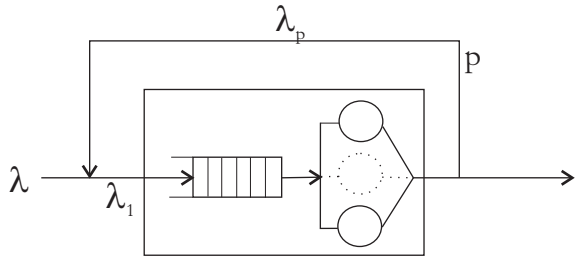
Rešitev: Izračuna bi bila sledeča:

$$(a) x = L/D, T = \frac{1}{\mu - \lambda}, N = \frac{\lambda}{\mu - \lambda}, \tag{3.117}$$

$$(b) T = L/D + \frac{\lambda * x^2}{2(1-\rho)}, N = x * \lambda + \frac{\lambda^2 * x^2}{2(1 - \lambda L/D)}. \tag{3.118}$$

3.14 Strežna enota z delnim vračanjem zahtev v strežbo

Shema tovrstnega sistema je predstavljena na sliki 3.15.



Slika 3.15: Strežna enota z delnim vračanjem zahtev v strežbo.

Z vidika strežne enote lahko zapišemo izraze, preko katerih pridemo do notranje intenzivnosti prihajanja λ_1:

$$\lambda_1 = \lambda + \lambda_p, \tag{3.119}$$

$$\lambda_p = \lambda_1 * p, \tag{3.120}$$

$$\lambda_1 = \frac{\lambda}{1-p}. \tag{3.121}$$

V splošnem v odvisnosti od arhitekture strežne enote v notranjosti strežnega sistema veljata izraza

$$T(\lambda_1) = T(\frac{\lambda}{1-p}), \tag{3.122}$$

$$T = \frac{T(\lambda_1)}{1-p} = \frac{T(\frac{\lambda}{1-p})}{1-p}. \tag{3.123}$$

3.15 Strežne mreže

Strežna mreža je sestavljena iz n strežnih enot. Za slednje smo rekli, da so sestavljene iz največ ene čakalne vrste in k paralelno vezanih ekvivalentno zmogljivih strežnikov. Strežne mreže ločimo na *odprte* in *zaprte*. V prve vstopajo zahteve iz zunanjega sveta in se vanj postrežene vračajo, v drugih pa kroži konstantno število zahtev, ki sistema ne zapuščajo. Poleg tega vanj nove zahteve ne vstopajo. Stanje v strežni mreže ponazarjamo s številom zahtev v posamezni strežni enoti ali z vektorjem populacije zahtev ($q(t) = (j_1, j_2, ..., j_n)$). Za zaprte strežne mreže veljata izraza

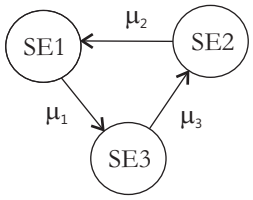
$$K = \sum_{i=1}^{n} j_i, \tag{3.124}$$

$$M = \binom{n+K-1}{n-1}, \tag{3.125}$$

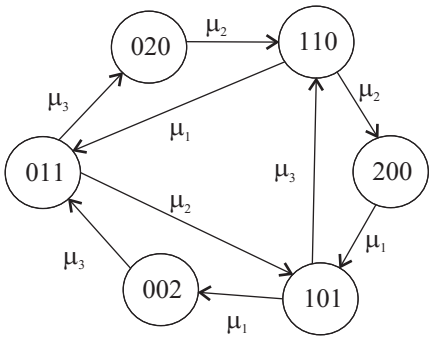
pri čemer je K število zahtev v zaprti strežni mreži, M pa število različnih možnih stanj mreže.

Zgled 5 *Predpostavimo, da imamo zaprto strežno mrežo s slike 3.16, v kateri sta dve zahtevi, vse strežne enote pa so tipa M/M/1. Kakšna je verjetnost stanja mreže z natanko eno zahtevo v prvi strežni enoti ob intenzivnostih strežbe posameznih strežnih enot μ_1, μ_2 in μ_3?*

Po izrazu (3.125) lahko izračunamo, da je $M = 6$. Na ta način pridemo do diagrama prehajanja med stanji, kot je prikazan na sliki 3.17.



Slika 3.16: Primer zaprte strežne mreže.



Slika 3.17: Diagram prehajanja stanj v zaprti strežni mreži.

Na osnovi diagrama prehajanja stanj nastavimo ravnotežne enačbe, kjer enačimo vhodne intenzivnosti in verjetnosti vhodnih stanj z izhodnimi intenzivnostmi opazovanega (izhodnega) stanja.

$$\mu_1 * p(2,0,0) = \mu_2 * p(1,1,0), \tag{3.126}$$

$$\mu_2 * p(0,2,0) = \mu_3 * p(0,1,1), \tag{3.127}$$

$$\mu_3 * p(0,0,2) = \mu_1 * p(1,0,1), \tag{3.128}$$

$$\mu_1 * p(1,1,0) + \mu_2 * p(1,1,0) = \mu_2 * p(0,2,0) + \mu_3 * p(1,0,1), \tag{3.129}$$

$$\mu_2 * p(0,1,1) + \mu_3 * p(0,1,1) = \mu_1 * p(1,1,0) + \mu_3 * p(0,0,2), \tag{3.130}$$

$$\mu_3 * p(1,0,1) + \mu_1 * p(1,0,1) = \mu_1 * p(2,0,0) + \mu_2 * p(0,1,1), \tag{3.131}$$

Ob upoštevanju dejstva, da je vsota verjetnosti nahajanja v enem od treh stanj 1, bi s substitucijami lahko izluščili verjetnosti $p(1,1,0)$ in $p(1,0,1)$. Vsota obeh bi dala odgovor na zastavljeno vprašanje.

3.16 Različne prioritete procesiranja

Poleg strežnih disciplin FIFO (angl. *first in first out*), LIFO (angl. *last in first out*), RND (naključno jemanje iz vrste), deljenja časovnih rezin (angl. *time sharing*), SJF (angl. *shortest job first*), LJF (angl. *largest job first*), v strežnem sistemu lahko predpostavimo tudi strežno disciplino na osnovi prioritet posameznih zahtev. Vstopajoče zahteve imajo tako lahko različne prioritete ($p = 1, ..., P$), pri čemer se iz vrste jemlje zahteve v vrstnem redu glede na njihovo prioriteto. Pri tem ločujemo med *neprekinitvenim* in *prekinitvenim* modelom prioritetnega procesiranja. Pri prvem se ne glede na vrsto novo prispele zahteve konča strežba obdelovane zahteve, potem pa se prevzame iz vrste zahtevo z najvišjo prioriteto, v drugem primeru pa se ob vstopu zahteve z višjo prioriteto, kot je prioriteta tiste, ki se obdeluje, obdelava prekine in prevzame se prispelo zahtevo z višjo prioriteto.

Za splošen M/G/1 sistem veljajo izrazi

$$\lambda = \sum_{p=1}^{P} \lambda_p, \;\; x = \sum_{p=1}^{P} \frac{\lambda_p}{\lambda} x_p, \tag{3.132}$$

$$\rho_p = \lambda_p * x_p, \;\; \rho = \lambda * x = \sum_{p=1}^{P} \rho_p, \tag{3.133}$$

$$T_p = W_p + x_p, \tag{3.134}$$

kjer indeks p pri posamezni spremenljivki določa njeno prioriteto, P pa največjo možno prioriteto zahteve.

3.16.1 Neprekinitveni model procesiranja

Za omenjeni model velja, da ne glede na prioriteto novo prispele zahteve, slednja ne more prekiniti že obstoječega servisiranja. Čas čakanja v vrsti novoprispele zahteve s prioriteto p je sestavljen iz naslednjih časov:

1. časa strežbe zahteve, ki se že nahaja v strežniku,
2. časa, da se postrežejo vse zahteve z višjo ali enako prioriteto iz čakalne vrste in
3. časa, da se postrežejo zahteve z višjimi prioritetami, ki so prispele v vrsto med čakanjem opazovane zahteve.

Povprečni čas strežbe iz točke 1. lahko določimo z izrazom

$$W_0 = \sum_{i=1}^{P} \rho_i \frac{\overline{x}_i^2}{2 * \overline{x}_i} = \sum_{i=1}^{P} \frac{\lambda_i \overline{x}_i^2}{2}. \tag{3.135}$$

Če ob tem uvedemo še spremenljivki N_{ip}, ki predstavlja število zahtev prioritete i, ki se bodo postregle pred opazovano zahtevo in M_{ip}, ki predstavlja število

zahtev, ki so vstopile kasneje in imajo višjo prioriteto, bi lahko čakalni čas v vrsti zahteve s prioriteto p zapisali z izrazom

$$W_p = W_0 + \sum_{i=1}^{P} \overline{x}_i (N_{ip} + M_{ip}). \tag{3.136}$$

Za zahtevo s prioriteto p tako veljajo naslednji izrazi

$$N_{ip} = 0 : i = 0, ..., p-1, \tag{3.137}$$

$$M_{ip} = 0 : i = 1, ..., p, \tag{3.138}$$

$$N_{ip} = \lambda_i W_i : i = p, ..., P, \tag{3.139}$$

$$M_{ip} = \lambda_i W_i : i = p+1, ..., P. \tag{3.140}$$

Na osnovi predhodnih izrazov lahko za W_p zapišemo nov izraz

$$W_p = W_0 + \sum_{i=p}^{P} \overline{x}_i \lambda_i W_i + \sum_{i=p+1}^{P} \overline{x}_i \lambda_i W_i, \tag{3.141}$$

za čas čakanja zahteve z največjo prioriteto pa

$$W_P = \frac{W_0}{1 - \rho_P}. \tag{3.142}$$

3.16.2 Prekinitveni model procesiranja

Za omenjeni model velja, da lahko novoprispela zahteva z višjo prioriteto, kot jo ima tista v servisiranju, prekine servisiranje in sama vstopi v strežnik. Tudi za ta primer bi lahko sestavili izraze po vzoru predhodnega razdelka, pti čemer smo v prejšnjem razdelku enačbe nastavili na osnovi časa čakanja v vrsti, v tem primeru pa bi jih nastavili na osnovi časa zadrževanja v strežni enoti.

Poglavje 4

Petrijeve mreže

4.1 Uvod

Petrijeve mreže predstavljajo univerzalno orodje za modeliranje in simulacijo dinamičnih sistemov. Na osnovi postavljenega modela in izvedene simulacije lahko pridemo do analize *dinamike* v sistemu, ki nam pove, ali je slednja v sistemu ustrezna ali ne. Pod pojmom dinamike v sistemu smatramo časovno sekvenco *stanj sistema*. Dinamičen sistem torej skozi čas spreminja svoja stanja. Analiza dinamike nam pomaga pri odločitvah ali je sistem potrebno popraviti, dopolniti, itd.

Temelje Petrijevih mrež je postavil Carl Adam Petri leta 1962. Definicijo njegovih mrež bomo v tem delu imenovali za *osnovne Petrijeve mreže* [5]. Poleg osnovnih Petrijevih mrež poznamo tudi *razširjene*. Mednje npr. sodijo *barvne*, *časovne*, *stohastične* in *mehke* Petrijeve mreže. Na področju računalništva se uporabljajo na področjih modeliranja komunikacijskih protokolov, analize zmogljivosti in zanesljivosti računalniških sistemov, analize verifikacije pravilnega delovanja algoritmov itd.

4.2 Gradniki Petrijevih mrež

Struktura Petrijeve mreže je sestavljena iz sledečih štirih tipov gradnikov:

- *pogojev* (angl. *places*),
- *akcij* (angl. *transitions*),
- *usmerjenih povezav* med akcijami in pogoji (ter obratno) in
- *žetonov*, s katerimi so definirane kratnosti izpolnjenosti posameznih pogojev.

Dinamiko v kakršnemkoli sistemu lahko interpretiramo kot zaporedje izvedenih akcij, za izvedbo katerih morajo biti izpolnjeni neki *predhodno* določeni

pogoji. Proženje posamezne akcije je torej pogojeno z vnaprej določeno množico izpolnjenih pogojev, rezultat proženja akcije pa povzroči izpolnjenost novega ali več novih pogojev. Slednje relacije so ponazorjene z usmerjenimi povezavami, ki vodijo iz pogojev v akcije in s povezavami, ki vodijo iz akcij v pogoje. Posamezen pogoj v Petrijevi mreži je lahko izpolnjen ali pa ne. V primeru izpolnjenosti pogoja govorimo o kratnosti izoplnjenosti pogoja. Tako je lahko pogoj izpolnjen enkrat, dvakrat ali večkrat, lahko pa ni izpolnjen. Žetone v Petrijevi mreži uporabljamo za označevanje kratnosti izpolnjenosti posameznih pogojev. En žeton v pogoju tako predstavlja enkratno izpolnjen pogoj, dva žetona dvakratno izpolnjen pogoj itd.

4.3 Osnovne Petrijeve mreže

Osnovne Petrijeve mreže v svoji definiciji ne predvidevajo časa trajanja akcij (čas trajanja akcij je hipen), niti ne predvidevajo stohastično pogojene dinamike. Informacijska vrednost žetona govori le o kratnosti izpolnjenosti pogoja, sam žeton pa ne nosi nobene druge dodatne informacijske vsebine. V nadaljevanju pričujočega razdelka bomo osnovnim Petrijevim mrežam rekli kar Petrijeve mreže.

4.3.1 Formalna definicija Petrijeve mreže

Zapišimo formalno definicijo Petrijeve mreže povzeto po viru [5].

Definicija 3 *Petrijeva mreža je definirana kot četvorček C=(P,T,I,O), pri čemer P predstavlja končno množico pogojev, T končno množico akcij, I vhodno in O izhodno funkcijo. Množici P in T sta si tuji ($P \cap T = \emptyset$).*

Vhodna in izhodna funkcija se nanašata na relacije med akcijami in pogoji. Vhodna funkcija I za akcijo t_j tako določa množico pogojev $I(t_j)$, iz katerih vodijo povezave proti akciji t_j, izhodna funkcija O pa za akcijo t_j določa množico pogojev $O(t_j)$, v katere vodijo povezave iz akcije t_j. Glede na povedano so možne povezave le iz akcij v pogoje in obratno, niso pa dovoljene povezave iz pogoja v pogoj ali iz akcije v akcijo. Običajno obe funkciji zapišemo v obliki prehajalnih matrik reda $m \times n$, pri čemer m predstavlja število akcij ($m = |T|$) in n število pogojev ($n = |P|$), ali v obliki *posplošenih* množic.

V izrazih (4.1) do (4.4) je predstavljen primer formalnega zapisa Petrijeve mreže s tremi akcijami, štirimi pogoji in desetimi povezavami najprej v matrični notaciji, nato pa še v notaciji posplošenih množic. Pri tem izraza (4.1) in (4.2) predstavljata *matrični način* formalne definicije, izrazi (4.1), (4.3) in (4.4) pa formalno definicijo na osnovi posplošenih množic.

$$C = (P, T, I, O), P = \{p_1, p_2, p_3, p_4\}, T = \{t_1, t_2, t_3\}, \tag{4.1}$$

$$I = \begin{bmatrix} 1 & 1 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{bmatrix}, \; O = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 2 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}. \tag{4.2}$$

$$I(t_1) = \{p_1, p_2, p_3\}, I(t_2) = \{p_4\}, I(t_3) = \{p_3\}, \tag{4.3}$$

$$O(t_1) = \{p_1\}, O(t_2) = \{p_2, p_2, p_3\}, O(t_3) = \{p_4\}. \tag{4.4}$$

V našem delu se bomo v nadaljevanju držali zgolj matrične notacije.

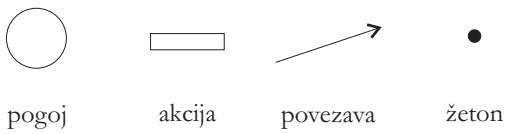
4.3.2 Definicija grafa Petrijeve mreže

Grafično ponazoritev formalno definirane Petrijeve mreže imenujemo *graf Petrijeve mreže*. Njegova definicija je po [5] sledeča:

Definicija 4 *Graf Petrijeve mreže je bipartitni usmerjeni graf G=(V,A), kjer V ($V = P \cup T$) predstavlja množico vozlišč in A množico povezav med njimi ($a_i = (v_j, v_k)$). Velja naslednja relacija:*

$$\forall i : a_i = (v_j, v_k),\ (v_j \in T)\&(v_k \in P) \vee (v_j \in P)\&(v_k \in T). \tag{4.5}$$

Za tvorbo grafov Petrijevih mrež uporabljamo grafične primitive, ki so prikazani na sliki 4.1, graf Petrijeve mreže, ki smo jo formalno zapisali v izrazih (4.1) in (4.2) pa na sliki 4.2.



Slika 4.1: Grafični primitivi za tvorbo grafa Petrijeve mreže.

4.3.3 Označitve pogojev v Petrijevih mrežah

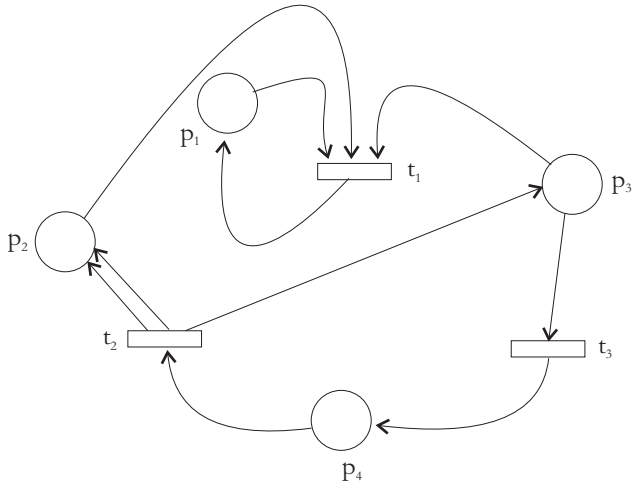
Kot smo povedali že v uvodu z označitvami pogojev v Petrijevih mrežah *določamo* ali *odčitavamo* kratnost izpolnjenosti pogojev. Formalno *označitev* Petrijeve mreže zapišemo kot vektor

$$o(k) = (o_1(k), o_2(k), ..., o_n(k)), \forall i : (o_i(k) \geq 0)\&(o_i(k) \in \mathbb{N} \cup \{0\}), i = 1, .., n, \tag{4.6}$$

pri čemer n predstavlja število pogojev ($n = |P|$), $o_i(k)$ pa število žetonov v i-tem pogoju ali kratnost izpolnjenosti i-tega pogoja. Skozi čas se v sistemu sprožajo različne akcije in načeloma se s tem spreminjajo tudi izpolnjenosti posameznih pogojev. Iz tega razloga je elementom vektorja kot tudi vektorju označitve dodan diskretni časovni atribut k. Oglejmo si še formalno definicijo označitve.

Definicija 5 *Označitev Petrijeve mreže C=(P,T,I,O) je funkcija, ki množico stanj P preslika v vektor nenegativnih celih števil.*

$$o : P \to \mathbb{N} \cup \{0\}. \tag{4.7}$$



Slika 4.2: Primer grafa Petrijeve mreže.

Z označitvijo vseh pogojev pridemo do *označene Petrijeve mreže*, ki jo zapišemo kot petorček $M = (P, T, I, O, o(k))$. Glede na to, da definicija označitve (4.6) ne predvideva omejitve števila žetonov v posameznem pogoju, za vsako Petrijevo mrežo obstaja neskončno možno označitev, če ima mreža vsaj en pogoj.

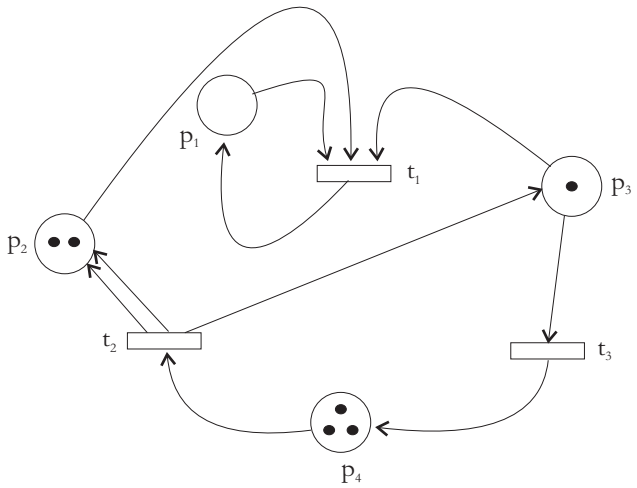
Za zgled mreže formalno definirane v izrazih od (4.1) do (4.4) predpostavimo začetno označitev $o(k_0) = (0, 2, 1, 3)$. V tem primeru bi prišli do grafične ponazoritve mreže, ki jo prikazuje slika 4.3. V primeru, da bi bilo žetonov v posameznem pogoju preveliko, bi v takšen pogoj zapisali število, ki bi predstavljalo kratnost izpolnjenosti pogoja. Označitev mreže $o(k)$ lahko enačimo s *stanjem*, v katerem se nahaja mreža na k-tem diskretnem koraku.

4.3.4 Proženje akcij v Petrijevih mrežah

Proženje posameznih akcij je pogojeno s trenutno označitvijo v Petrijevi mreži. Posamezna akcija se lahko sproži, če je *omogočena*. Akcija t_i je omogočena, če se po vsaki povezavi, ki vstopa v to akcijo, lahko pripelje žeton iz pogoja, ki predstavlja izvor povezave (vhodni pogoj). To lahko formalno zapišemo z izrazom

$$o(k) \geq e(t_i) * I, \ i = 1, ..., m, \tag{4.8}$$

pri čemer je $e(t_i)$ enotski vektor dolžine m ($m = |T|$), enica pa se nahaja na i-ti poziciji, ki jo definira indeks akcije. Sprožitev akcije torej v splošnem odvzema žetone iz vhodnih pogojev - pogojev iz katerih vodijo povezave proti opazovani akciji t_i. Po hipnem trajanju akcije (predpostavljamo, da je čas trajanja ničen ali zanemarljivo majhen), pride do posledičnega izražanja na izhodnih pogojih - pogojih do katerih vodijo izhodne povezave iz opazovane akcije t_i. Praviloma se



Slika 4.3: Primer označenega grafa Petrijeve mreže.

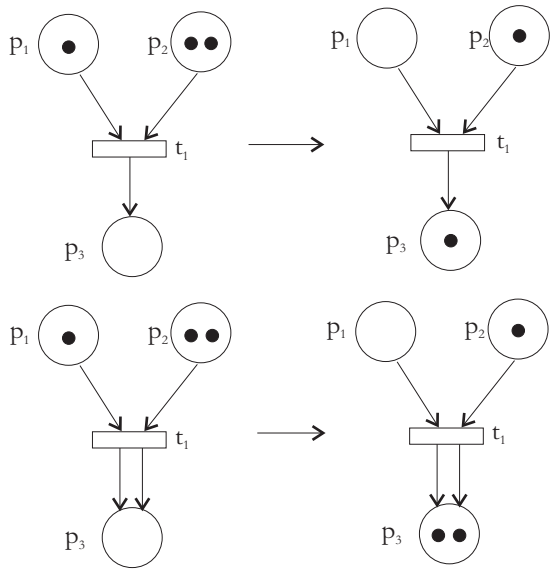
po hipnem proženju akcije t_i odpravi po vsaki izhodni povezavi proti izhodnim pogojem en žeton. Vsak od njih se uskladišči v izhodnem pogoju. Prehajanje žetonov od opazovane - prožene akcije t_i proti izhodnim pogojem formalno in s tem posledično spremenjeno označitev sistema $o(k+1)$ zapišemo z izrazom

$$o(k+1) = o(k) + e\,[t_i]\,(O - I). \tag{4.9}$$

Posebno je potrebno poudariti, da je potovanje posameznega žetona od vhodnega pogoja preko prožene akcije do izhodnega pogoja hipno (brez časovnega trajanja). V nadaljevanju si oglejmo nekaj konkretnih zgledov dinamik, ki so predstavljene na sliki 4.4. Iz slike je razvidno, da ni nujno, da je število žetonov, ki vstopijo v akcijo, enako številu žetonov, ki iz nje izstopijo.

Za primer povzemimo mrežo definirano formalno z matrikama I in O v izrazu (4.10) s slike 4.5 ob začetni označitvi $o(t) = (1, 0, 0, 0)$. Glede na začetno označitev so omogočene akcije t_1, t_2 in t_3. Ob predpostavki, da se istočasno ne moreta sprožiti dve akciji, imamo tri možne toke dogajanj, ki jih bomo ponazorili z *drevesom označitev*, predstavljenim na sliki 4.6. Do drevesa označitev lahko pridemo z izračunavanjem po izrazih (4.8) in (4.9), ali pa s postopnim prestavljanjem žetonov po grafu Petrijeve mreže. Izračun za levo vejo prehajalnega drevesa je predstavljen v izrazih od (4.12) do (4.15).

$$O = \begin{bmatrix} 0 & 1 & 0 & 0 \\ 0 & 1 & 1 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 \end{bmatrix}, \; I = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 2 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{4.10}$$



Slika 4.4: Zgledi dinamike žetonov v Petrijevih mrežah.

$$O - I = \begin{bmatrix} -1 & 1 & 0 & 0 \\ -1 & 1 & 1 & 0 \\ -1 & 0 & 1 & 0 \\ 0 & -1 & 1 & 0 \\ 0 & 0 & -2 & 1 \\ 1 & 0 & 0 & -1 \end{bmatrix} \tag{4.11}$$

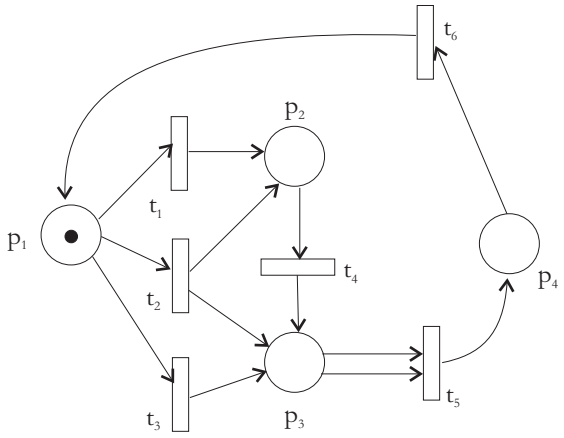
$$t_1 : o(t+1) = (1,0,0,0) + (1,0,0,0,0,0) * \begin{bmatrix} -1 & 1 & 0 & 0 \\ -1 & 1 & 1 & 0 \\ -1 & 0 & 1 & 0 \\ 0 & -1 & 1 & 0 \\ 0 & 0 & -2 & 1 \\ 1 & 0 & 0 & -1 \end{bmatrix} = \tag{4.12}$$

$$= (1,0,0,0) + (-1,1,0,0) = (0,1,0,0), \tag{4.13}$$

$$t_4 : o(t+2) = (0,1,0,0) + (0,0,0,1,0,0) * \begin{bmatrix} -1 & 1 & 0 & 0 \\ -1 & 1 & 1 & 0 \\ -1 & 0 & 1 & 0 \\ 0 & -1 & 1 & 0 \\ 0 & 0 & -2 & 1 \\ 1 & 0 & 0 & -1 \end{bmatrix} = \tag{4.14}$$

$$= (0,1,0,0) + (0,-1,1,0) = (0,0,1,0) \tag{4.15}$$

Ob analizi proženja akcij moramo biti pozorni na akcije, ki nimajo vhodnih pogojev. Tovrstne akcije so vedno omogočene. Tipičen primer takšne akcije je



Slika 4.5: Primer grafa Petrijeve mreže za izdelavo drevesa označitev.

predstavljen na sliki 4.7. Akcije, ki se lahko ves čas prožijo, tvorijo v drevesu označitev *neskončne veje*. Poleg neskončnih vej moramo biti v drevesu pozorni na *končna stanja* (liste drevesa), kjer se proženje akcij ustavi in na stanja, ki se skozi dinamiko ponavljajo (*ciklična stanja*).

S proženjem akcij v Petrijevi mreži se porajata dve vrsti sekvenc. Prva je sekvenca označitev, druga pa sekvenca akcij. Stanje Petrijeve mreže se odslikuje s trenutno označitvijo.

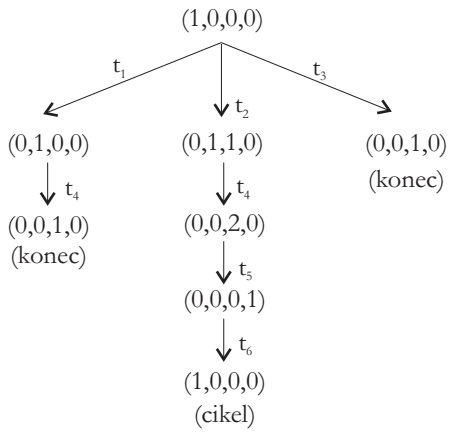
Označitev o' je *neposredno dosegljiva* iz označitve o, ko obstaja takšna akcija t_j, ki nas z izvajanjem pripelje iz označitve o v označitev o'. Povedano drugače je o' neposredno dosegljiva takrat, ko velja $\delta(o, t_j) = o'$, pri čemer δ predstavlja zapis preslikovalne funkcije v novo označitev (stanje).

Množica *dosegljivih stanj* $R(C, o)$ je množica vseh označitev (stanj Petrijeve mreže), ki so dosegljiva iz označitve o. Označitev o' pripada množici $R(C, o)$, če obstaja zaporedje akcij, ki nas pripelje iz označitve o v označitev o'. Velja naslednja relacija:

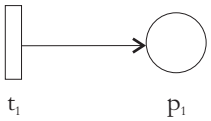
$$if\ (o' \in R(C,o))\ and\ (\exists t_j \in T : o'' = \delta(o', t_j))\ then\ o'' \in R(C,o). \qquad (4.16)$$

4.4 Zgledi modeliranja s Petrijevimi mrežami

V pričujočem razdelku si bomo ogledali nekaj konkretnih primerov modeliranja različnih sistemov s Petrijevimi mrežami. Večina primerov je povzeta po viru [5].



Slika 4.6: Drevo označitev za primer Petrijeve mreže s slike 4.5.



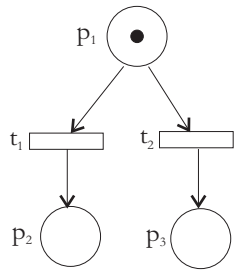
Slika 4.7: Primer akcije brez vhodnih pogojev, ki se neprestano proži.

4.4.1 Nedeterminizem in konkurenčnost v Petrijevih mrežah

Na sliki 4.8 je predstavljen graf Petrijeve mreže, ki ponazarja *nedeterminističnost* izbire poti žetona in *konkurenčnost* akcij. V kontekstu prvega pojma se žeton nedeterministično odloča o izbiri akcije, v kontekstu drugega pojma pa smatramo konkurenco ali kompeticijo med akcijama t_1 in t_2, ki skušata za svojo izvedbo pridobiti žeton.

4.4.2 Problem dveh kitajskih meditatorjev

Dva meditatorja sedita za okroglo mizo ter se izmenično prehranjujeta in meditirata. Za prehranjevanje potrebuje posamezni meditator dve paličici. Na mizi sta na začetku dve paličici za prehranjevanje s kitajsko hrano. Ena je na levi in ena na desni med sedočima, tako da imata oba občutek, da imata dostop do obeh paličic, ki jih nujno potrebujeta za prehranjevanje. Seveda se lahko glede na število razpoložljivih paličic prehranjuje le eden. Na sliki 4.9 je predstavljen graf Petrijeve mreže, ki ponazarja omenjeno situacijo. Predstavljena rešitev podpira nedeterministično izbiro meditatorja pri dodelitvi dveh paličic naenkrat. Navedimo pomene posameznih pogojev in akcij:



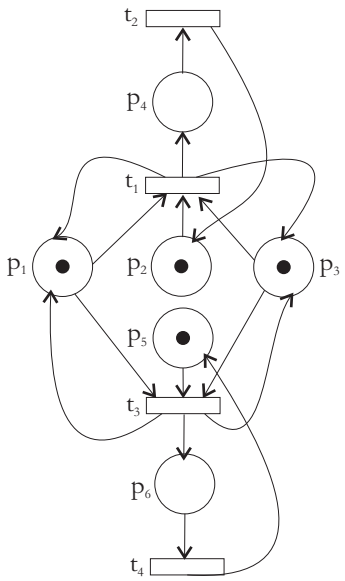
Slika 4.8: Graf Petrijeve mreže nedeterministične izbire poti žetona in konkurenčnosti dveh akcij.

- p_1: prva paličica je prosta,
- p_2: prvi meditator je pripravljen za prehranjevanje,
- p_3: druga paličica je prosta,
- p_4: prvi meditator je pripravljen za meditacijo,
- p_5: drugi meditator je pripravljen za prehranjevanje,
- p_6: drugi meditator je pripravljen za meditacijo,
- t_1: prvi meditator se prehranjuje,
- t_2: prvi meditator meditira,
- t_3: drugi meditator se prehranjuje,
- t_4: drugi meditator meditira.

4.4.3 Računalniški strežni sistem

Računalniški strežni sistem sestavljajo računalniki R1, R2 in R3. Vsaka zahteva, ki vstopi v sistem, mora biti najprej obdelana na R1, nato pa na R2 ali R3. Na sliki 4.10 je predstavljen graf Petrijeve mreže, ki ponazarja omenjeno situacijo. Rešitev podpira nedeterministično izbiro vira (resursa) R2 ali R3. Povezave narisane s prekinjenimi črtami označujejo vhodno in izhodno točko strežnega sistema. Navedimo pomene posameznih pogojev in akcij:

- p_1: v sistem je vstopila zahteva,
- p_2: računalnik R1 je prost,
- p_3: zahteva čaka na obdelavo v drugi fazi,
- p_4: zahteva je dokončno sprocesirana,



Slika 4.9: Graf Petrijeve mreže za problem dveh kitajskih meditatorjev.

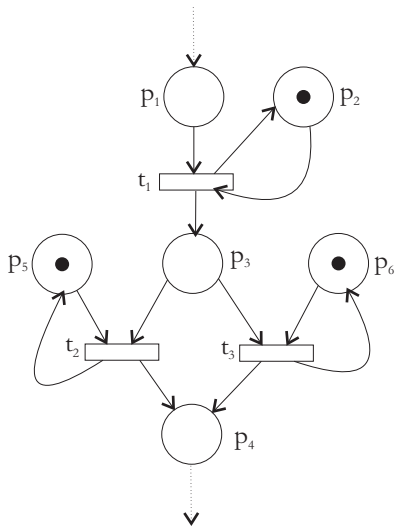
- p_5: računalnik R2 je prost,
- p_6: računalnik R3 je prost,
- t_1: zahteva se obdeluje na R1,
- t_2: zahteva se obdeluje na R2,
- t_3: zahteva se obdeluje na R3.

4.4.4 DO WHILE programski stavek s čuvajem

DO WHILE programski stavek definira izvajanje programskega stavka S, pri čemer se ta prvič izvede brezpogojno, vse nadaljnje izvedbe pa so pogojene z veljavnostjo pogoja P. Formalno ga lahko zapišemo z izrazom

$$do\ \{S\}\ while(P). \tag{4.17}$$

Na sliki 4.11 je predstavljen graf Petrijeve mreže, ki ponazarja opisani sistem. Povezava zaključena s krožcem (p_5-t_3) predstavlja njeno *inhibirno* naravo. Slednje pomeni, da se izhodna akcija t_3 lahko sproži le pod pogojem, da v izvornem pogoju inhibirne povezave ni žetonov. Na tem mestu omenimo, da inhibirna povezava predstavlja razširjavo klasičnih Petrijevih mrež. Nenazadnje nam ob njeni vpeljavi ne služita več izraza (4.8) in (4.9).

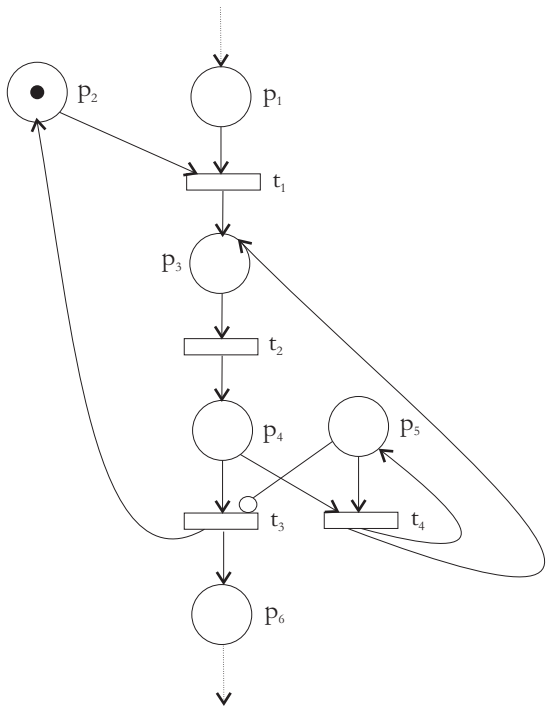


Slika 4.10: Graf Petrijeve mreže za ponazoritev računalniškega strežnega sistema.

Pogoj p_2 predstavlja funkcijo *čuvaja*, ki onemogoča vstope v DO WHILE sekvenco, če je le ta že v uporabi. Navedimo pomene posameznih pogojev in akcij:

- p_1: v sistem je vstopila zahteva za izvajanje prog.stavka,
- p_2: predstavlja funkcijo čuvaja, ob prisotnosti katerega je možno izvršiti zanko,
- p_3: zahteva čaka na izvedbo S stavka,
- p_4: izvedba S stavka je opravljena,
- p_5: pogoj P,
- t_1: izvede se vstop v sekvenco (ob prisotnosti čuvaja),
- t_2: izvede se S stavek,
- t_3: ker pogoj P ni izpolnjen, se izvajanje zaključi in čuvaj se vrne v razpoložljivo stanje,
- t_4: ker pogoj P je izpolnjen, se vrnemo na ponovno brezpogojno izvajanje stavka S.

Kot smo povedali, p_5 predstavlja preverjani pogoj P. Akcija t_4 se izvaja vse dotlej, dokler je v p_5 kakšen žeton (dokler je pogoj izpolnjen). Ker s tem t_4 odvzema žetone iz p_5 in s tem „ruši“ pogoj, vpeljemo povezavo $t_4 - p_5$, ki žetone p_5 vrača. Tako samo preverjanje pogoja P slednjega ne spreminja.



Slika 4.11: Graf Petrijeve mreže za ponazoritev DO WHILE stavka.

4.4.5 IF programski stavek s čuvajem

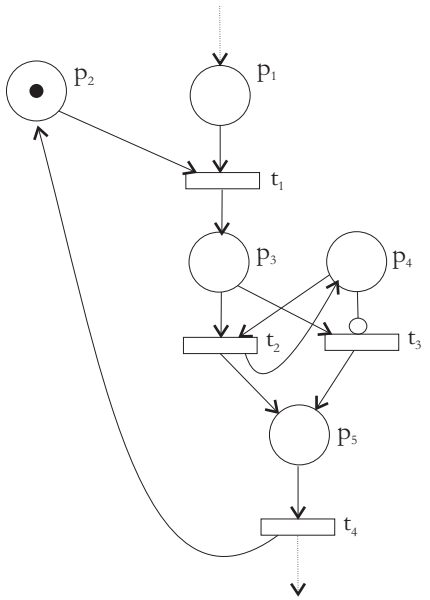
IF programski stavek definira izvajanje programskega stavka S_1, če je izpolnjen pogoj P, v nasprotnem primeru pa izvajanje stavka S_2. Formalno ga lahko zapišemo z izrazom

$$if\ (P)\ then\ \{S_1\}\ else\ \{S_2\}\,. \tag{4.18}$$

Na sliki 4.12 je prikazan graf Petrijeve mreže, ki ponazarja omenjeni sistem. Pogoj p_2 zopet predstavlja funkcijo čuvaja, povezava p_4 - t_3 pa nestandardno inhibirno povezavo. Navedimo še ostale pomene posameznih pogojev in akcij:

- p_1: v sistem je vstopila zahteva za izvajanje IF stavka,
- p_2: predstavlja funkcijo čuvaja,
- p_3: zahteva je pred odločitvijo, kateri stavek izvesti,
- p_4: predstavlja pogoj P,
- p_5: eden od stavkov je izveden,
- t_1: izveden je vstop v stavek,

- t_2: izvede se S_1,
- t_3: izvede se S_2,
- t_4: stavek je dokončno izveden.



Slika 4.12: Graf Petrijeve mreže za ponazoritev IF stavka.

4.4.6 FOR programski stavek s čuvajem

FOR programski stavek predvideva izvajanje programskega stavka S v n ponovitvah. Formalno ga lahko zapišemo z izrazom

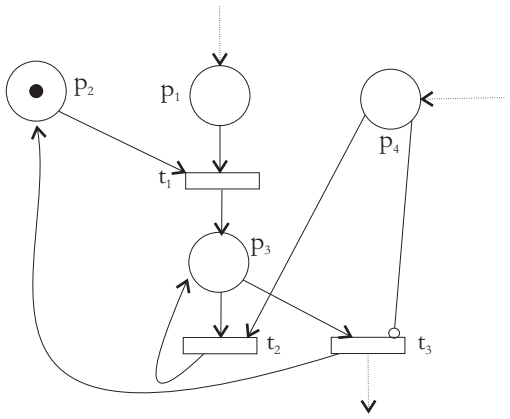
$$for\ (i = 1\ to\ n)\ do\ \{S\}\,. \tag{4.19}$$

Na sliki 4.13 je prikazan graf Petrijeve mreže, ki ponazarja omenjeni sistem. Navedimo pomene posameznih pogojev in akcij:

- p_1: v sistem je vstopila zahteva za izvajanje FOR stavka,
- p_2: predstavlja funkcijo čuvaja,
- p_3: zahteva je pred odločitvijo ali stavek S izvesti ali ne,
- p_4: predstavlja števec n, ki mora biti predhodno inicializiran z n žetoni,
- t_1: izveden je vstop v stavek,

- t_2: izvede se stavek S,
- t_3: FOR stavek je dokončno izveden v celoti.

Črtkana povezava, ki vstopa v p_4 brez izvora, nas opozarja na to, da moramo pred samo izvedbo FOR stavka inicializirati število žetonov (število n). Ključni točki stavka sta v akcijah t_2 in t_3. Prva se izvaja vse dotlej, dokler je v p_4 še kakšen žeton ($n>0$), ko pa le teh zmanjka ($n = 0$), se izvede akcija t_3.



Slika 4.13: Graf Petrijeve mreže za ponazoritev FOR stavka.

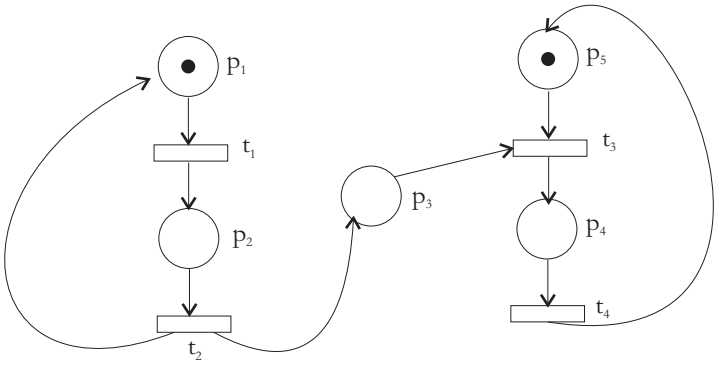
4.4.7 Proizvodno porabniški problem

Problem proizvajalca in porabnika (angl. *producer - consumer problem*) na eni strani definira proces proizvodnje, na drugi strani pa proces porabe. Na sliki 4.14 je predstavljen graf Petrijeve mreže, ki ponazarja omenjeno situacijo. Navedimo pomene posameznih pogojev in akcij:

- p_1: proizvodni proces je pripravljen za proizvodnjo ene enote,
- p_2: enota je proizvedena,
- p_3: funkcija vmesnika (angl. *buffer*), kjer proizvedene enote čakajo na porabo,
- p_4: enota je prevzeta iz vmesnika,
- p_5: proces porabe je pripravljen na porabo ene enote,
- t_1: proizvede se enota,
- t_2: proizvodni proces preda enoto v vmesnik in inicializira se pripravljenost na ponovno proizvodnjo (žeton preide v p_1),

- t_3: proces porabe prevzame enoto iz vmesnika,
- t_4: porabi se enota in inicializira ponovna pripravljenost na porabo (žeton preide v p_5).

Pomembno je, da začetna označitev inicializira pogoja p_1 in p_5.



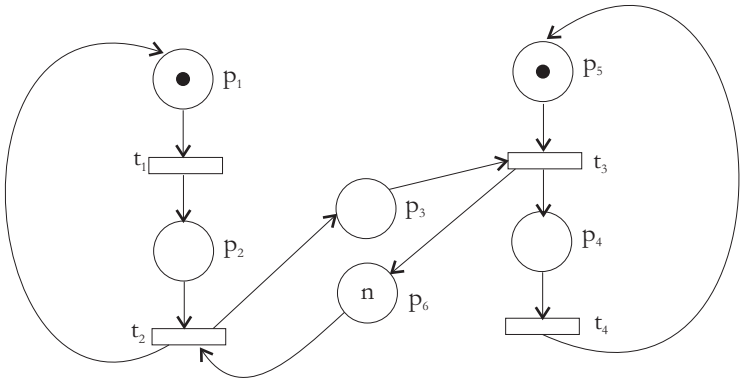
Slika 4.14: Graf Petrijeve mreže za ponazoritev proizvodno porabniškega sistema.

Omenjeni sistem ni najbolj realističen, ker lahko pride do prekoračitve kapacitete realiziranega vmesnika, ki praviloma nikdar nima neskončne kapacitete. V tem smislu na sliki 4.15 predstavimo dopolnitev predhodnega grafa Petrijeve mreže, ki omejuje prekomerno polnjenje vmesnika. V njem nastopa nov pogoj p_6, v katerem je na začetku n žetonov, pri čemer n predstavlja kapaciteto vmesnika. Ko se iz p_6 odstrani vse žetone, akcija t_2 ne more več odlagati enot v vmesnik in je potrebno za nadaljevanje procesa proizvodnje počakati fazo porabe, da p_6 napolni z vsaj enim žetonom (odvzame iz vmesnika vsaj eno enoto).

4.4.8 Problem branja in pisanja

Problem branja in pisanja (angl. *read - write problem*) je aktualen pri kakršnemkoli paralelnem dostopanju do podatkov. Pod pojmom dostopanja ločujemo med procesoma branja in pisanja. V primeru branja načeloma lahko omogočamo n paralelnih dostopov (pravic do branja), pri pisanju pa moramo predhodno podatke zakleniti in s tem onemogočiti vsa aktivna branja in morebitna ostala pisanja. Na sliki 4.16 je predstavljen graf Petrijeve mreže, ki ponazarja omenjeno situacijo. Navedimo pomene posameznih pogojev in akcij:

- p_1: pripravljena je zahteva za branje,
- p_2: omogočen je dostop do branja podatka,
- p_3: vmesnik, v katerem je shranjenih n dostopnih pravic,



Slika 4.15: Graf Petrijeve mreže za ponazoritev realnega proizvodno porabniškega sistema.

- p_4: omogočen je dostop do pisanja,
- p_5: pripravljena je zahteva za pisanje,
- t_1: izvede se zaseganje enkratne pravice za branje,
- t_2: izvede se branje in enkratna pravica do branja se vrne v p_3,
- t_3: izvede se zaseganje n kratne pravice za pisanje (zaseg n žetonov),
- t_4: izvede se pisanje in n kratna pravica za dostop se vrne v p_3.

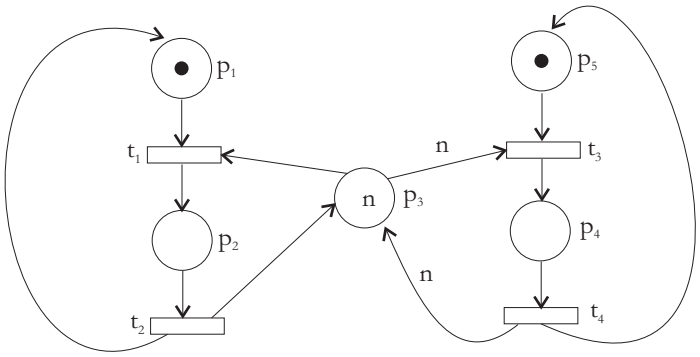
Z vidika začetne označitve je pomembna prisotnost vsaj enega žetona v levem (bralnem) ciklu, prisotnost vsaj enega žetona v desnem (pisalnem ciklu) in označitev števila pravic (p_3).

Posebno pozornost gre nameniti povezavama $p_3 - t_3$ in $t_4 - p_3$, ki sta n - kratni povezavi, kar pomeni, da se po takšni povezavi prelije n žetonov hkrati.

4.4.9 Problem medsebojnega izključevanja

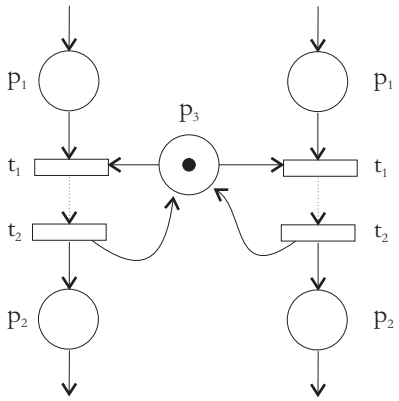
Problem medsebojnega izključevanja (angl. *mutual exclusion problem*) opozarja na možnost obstoja *kritičnih sekcij* sistema, ki ne smejo biti zasedene istočasno. Na sliki 4.17 je prikazan graf Petrijeve mreže, ki ponazarja model omenjenega problema s podeljevanjem pravice do dostopa do sekcije (stražarja). Z vidika začetne označitve je pomembna prisotnost žetona v p_3. Navedimo pomene posameznih pogojev in akcij:

- p_1: do kritične sekcije pristopi zahteva,
- p_2: zahteva zapusti kritično sekcijo,
- p_3: prisotnost enkratne pravice dostopa do kritične sekcije,



Slika 4.16: Graf Petrijeve mreže za ponazoritev problema branja in pisanja.

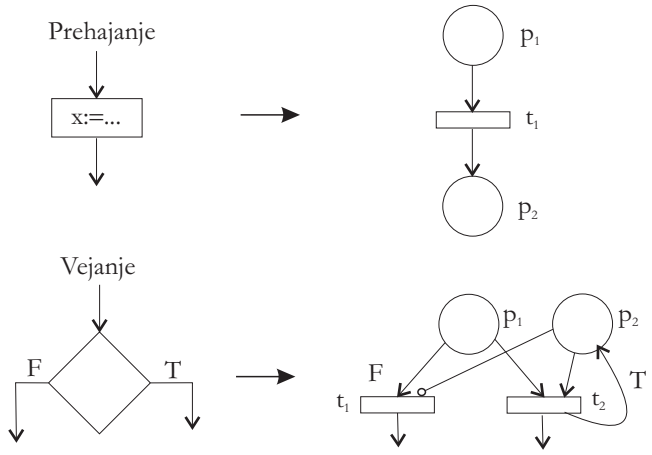
- t_1: dodelitev pravice za dostop do kritične sekcije in njen začetek,
- t_2: konec kritične sekcije in vračanje pristopne pravice v p_3.



Slika 4.17: Graf Petrijeve mreže za ponazoritev eliminacije možnosti istočasnega nahajanja zahtev v kritičnih sekcijah.

4.4.10 Modeliranje diagrama poteka

S Petrijevimi mrežami lahko modeliramo tudi diagrame poteka, ki so ena od osnovnih metod za enolično ponazarjanje algoritmov. Osnovna konstrukta diagramov poteka sta *prehajanje* in *vejanje*. Na sliki 4.18 najdemo ponazoritve, v kakšne Petrijeve konstrukte se preslikajo aktivnosti in v kakšne vejanja. Pri tem pogoj p_2 pri vejanju predstavlja preverjani pogoj, na osnovi katerega se vejanje vrši.



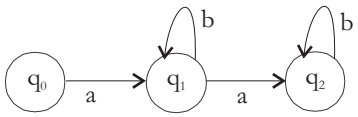
Slika 4.18: Prehod iz gradnikov diagramov poteka na osnovne konstrukte Petrijevih mrež.

vhodna črka - stanje	q_0	q_1	q_2
a	q_1 (t_{11})	q_2 (t_{12})	- (t_{13})
b	- (t_{21})	q_1 (t_{22})	q_2 (t_{23})

Tabela 4.1: Tabela prehajanj v avtomatu s slike 4.19.

4.4.11 Prehod med končnim avtomatom in Petrijevo mrežo

Predpostavimo, da imamo opravka s končnim avtomatom, predstavljenim na sliki 4.19. Omenjeni avtomat lahko ponazorimo s prehajalno tabelo 4.1.



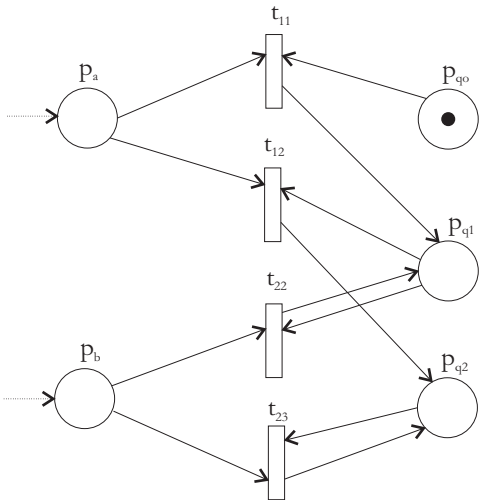
Slika 4.19: Primer končnega avtomata.

Pravila za formiranje ekvivalentne Petrijeve mreže lahko strnemo v naslednje alineje:

- črke vhodne abecede se preslikajo v pogoje (za naš primer tako velja $X = \{a, b\} \Rightarrow P_1 = \{p_a, p_b\}$),
- stanja avtomata se preslikajo v pogoje (za naš primer tako velja $Q = \{q_0, q_1, q_2\} \Rightarrow P_2 = \{p_{q_0}, p_{q_1}, p_{q_2}\}$),

- prehajanja med stanji se preslikajo v akcije (za na naš primer tako velja $T = \{t_{11}, t_{12}, t_{22}, t_{23}\}$),
- formira se množica pogojev P ($P = P_1 \cup P_2$) (za naš primer tako velja $P = \{q_0, q_1, q_2, p_a, p_b\}$).

Na sliki 4.20 je predstavljen graf Petrijeve mreže, ki je ekvivalentna avtomatu s slike 4.19. Z vidika izvajanja mreže je pomembno, da v enega od pogojev v mreži, ki ponazarjajo stanja (q_0, q_1, q_2), vstavimo en žeton, s čimer inicializiramo začetno stanje avtomata.



Slika 4.20: Pretvorba končnega avtomata s slike 4.19 v graf Petrijeve mreže.

4.4.12 Petrijeve mreže kot generatorji jezikov

V predhodnih razdelkih smo bili pozorni predvsem na pogoje, označitve in sekvence označitev. V pričujoče razdelku bomo pozornost preusmerili na sekvenco proženih akcij, ki okarakterizirajo modelirani sistem. V kontekstu jezikov Petrijevih mrež posamezna sprožena akcija predstavlja *znak*, sekvenca sproženih akcij *besedo*, vse možne sekvence proženih akcij pa *jezik* Petrijeve mreže. Ob tem smo že predhodno predpostavljali, da se dve akciji ne moreta sprožiti istočasno. Dve Petrijevi mreži sta *ekvivalentni*, če imata enake jezike. V nadaljevanju zapišimo formalno definicijo jezika.

Definicija 6 *Jezik L je jezik Petrijeve mreže, če obstajajo Petrijeva mreža $C=(P,T,I,O)$, označitev akcij $\rho : T \to \Sigma$, začetna označitev $o(t_0)$ in končna množica končnih označitev F, tako da velja:*

$$L = \{\rho(\beta) \in \Sigma^* : \beta \in T^* \text{ and } \delta(o(t_0), \beta) \in F\}. \tag{4.20}$$

Pri tem Σ predstavlja množico vseh črk jezika, Σ^* množico vseh možnih tvorb besed glede na množico črk jezika, F množico vseh besed jezika, β poljubno zaporedje akcij, δ pa preslikovalno funkcijo, ki na osnovi začetne označitve $o(t_0)$ in zaporedja akcij β formira veljavno besedo jezika.

Oglejmo si primer jezika $L = L_1 \cup L_2, L_1 = a(a \vee b)b, L_2 = a^n b^n,\ n \geq 1$. Rešitev generatorja jezika v obliki grafa Petrijeve mreže podajamo na zaporedju slik 4.21 (L_1 in L_2) in 4.22 ($L_1 \cup L_2$). Začetna inicializacija ni potrebna na nobeni od obeh navedenih slik. Na tem mestu povdarimo, da se tako izbira med tipom besede, kot tudi izbira dolžine sekvence n vršita nedeterministično.

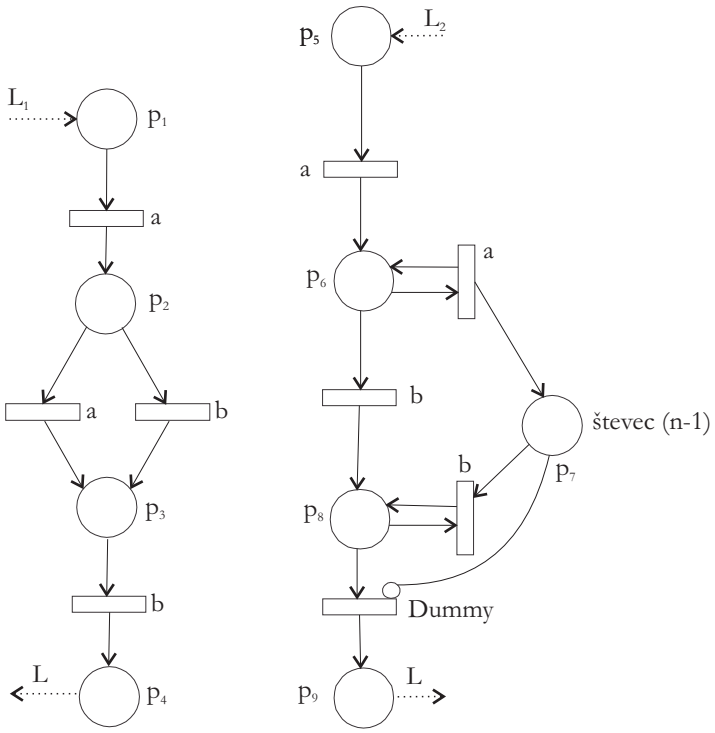
Opišimo še pomene pomembnejših pogojev in akcij:

- p_1: v sistem L_1 je vstopila zahteva za generiranje besede jezika L_1,
- p_2: nedeterministično odločanje med generiranjem črke „a“ ali „b“,
- p_4: beseda jezika L_1 je bila uspešno generirana,
- p_5: v sistem L_2 je vstopila zahteva za generiranje besede jezika L_2,
- p_6: izhodišče za nedeterministično generiranje $(n-1)$ pojavitev črke „a“,
- p_7: pomnjenje števila $(n-1)$ generiranj črke „a“,
- p_8: izhodišče za deterministično generiranje $(n-1)$ pojavitev črke „b“,
- p_9: beseda jezika L_2 je bila uspešno generirana,
- p_{11}: v sistem L je vstopila zahteva za generiranje besede jezika L in pride do nedetrminističnega izbiranja besede iz jezika L_1 ali L_2,
- p_{12}: beseda jezika L je bila uspešno generirana,
- Dummy: vse akcije s to oznako nimajo zmožnosti generiranja črke,
- a: akcije s to oznako generirajo črko „a“,
- b: akcije s to oznako generirajo črko „b“,.

Na sliko 4.22 bi bilo umestno dodati tudi čuvaja, ki bi zagotavljal, da ne bi v sistem (kritično sekcijo) vstopilo več zahtev za generiranje, kar bi vodilo do neustrezne sekvence akcij in posledično do odčitavanja nepravih besed jezika.

4.5 Analiza Petrijevih mrež

Rezultati modeliranja in simulacij s Petrijevimi mrežami nas vodita do možnosti analize dinamike v Petrijevih mrežah. Predhodno smo se že spoznali z metodo analize s pomočjo *drevesa dosegljivosti*, v nadaljevanju pa se bomo seznanili še z analitičnimi ocenami lastnosti *varnosti*, *omejenosti* in *konservativnosti*.



Slika 4.21: Primer Petrijevih mrež - generatorjev jezika L_1 (levo) in L_2 (desno).

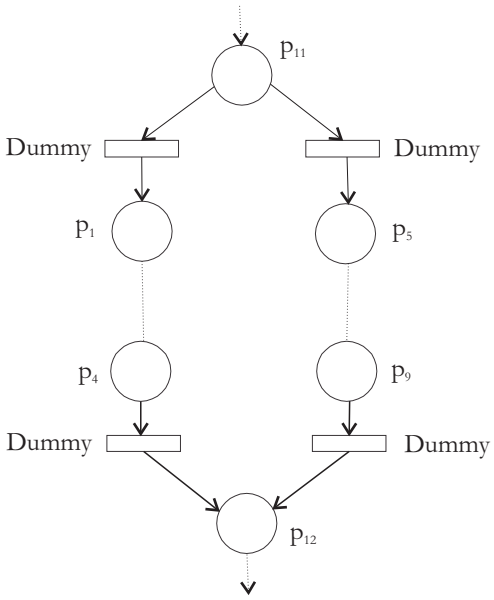
4.5.1 Varnost Petrijeve mreže

Ena od možnih značilnosti opazovane Petrijeve mreže je njena *varnost* (angl. *safeness*). Posamezen pogoj v mreži je *varen*, če se skozi simulacijo število žetonov v tem pogoju nikdar ne dvigne nad 1. Petrijeva mreža kot celota je *varna*, če so v njej varni tudi vsi pogoji. Zapišimo definicijo še formalno.

Definicija 7 *Pogoj $p_i \in P$ Petrijeve mreže C=(P,T,I,O) z začetno označitvijo o je varen, če za vse o'$\in R(C,o)$ velja o'$(p_i) \leq 1$. Petrijeva mreža C je varna, če so v njej varni vsi pogoji.*

Omenjena značilnost opazovane Petrijeve mreže je dobrodošla predvsem na področju modeliranja logičnih enačb, logičnih preklopnih struktur in podobnih sistemov, ki lahko zasedajo le dve možni stanji. Na ta način posamezni pogoj v Petrijevi mreži obravnavamo kot dvovrednostni logični pogoj.

V primeru, da pogoj p_i v Petrijevi mreži ni varen in nima tako večkratnih vhodnih, kot tudi večkratnih izhodnih povezav, obstaja način, da tudi ta pogoj spremenimo v varen pogoj. To dosežemo tako, da vpeljemo nov pogoj p_i'. Postopek vpeljave novega pogoja je zapisan v izrazih



Slika 4.22: Primer Petrijeve mreže - generatorja jezika $L_1 \cup L_2$.

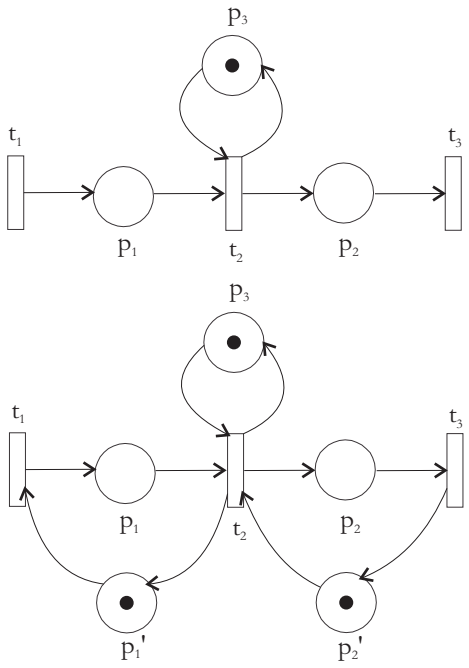
$$if\ (p_i \in I(t_j))\ and\ (p_i \notin O(t_j))\ then\ add\ p_i'\ to\ O(t_j), \tag{4.21}$$
$$if\ (p_i \in O(t_j))\ and\ (p_i \notin I(t_j))\ then\ add\ p_i'\ to\ I(t_j), \tag{4.22}$$

razložimo pa ga lahko na naslednji način. Če je pogoj p_i, ki ni varen, vhodni pogoj za akcijo t_j, ni pa njen izhodni pogoj, uvedemo nov izhodni pogoj p_i' akciji t_j. če pa je pogoj p_i, ki ni varen, izhodni pogoj za akcijo t_j, ni pa njen vhodni pogoj, uvedemo za akcijo t_j nov vhodni pogoj p_i'. Pogoj p_i' je komplementaren pogoju p_i, pomensko pa predstavlja izjavo, da je pogoj p_i neveljaven (v p_i ni žetonov). Po vpeljavi novega pogoja moramo poskrbeti tudi za spremembo začetne označitve, ki mora vsebovati žeton bodisi v p_i ali v p_i'. Na sliki 4.23 je prikazan primer pretvorbe pogojev p_1 in p_2 v varna pogoja.

Lastnost varnosti je zanimiva iz vidika pomnjenja, saj nam zniža število možnih stanj sistema in s tem tudi poceni realizacijo modela v obliki strojne realizacije.

4.5.2 Omejenost Petrijeve mreže

Omejenost je posplošena značilnost varnosti. Petrijeva mreža je k-omejena, če se število žetonov v posameznih pogojih skozi simulacijo nikdar ne dvigne čez mejo k, jo pa vsaj občasno in v vsaj nekaterih pogojih dosega.



Slika 4.23: Primer nevarne Petrijeve mreže (zgoraj) in njene pretvorbe v varno (spodaj).

Definicija 8 *Pogoj $p_i \in P$ Petrijeve mreže C=(P,T,I,O) z začetno označitvijo o je k-omejen, če za vse o'$\in R(C,o)$ velja o'$(p_i) \leq k$. Petrijeva mreža C je k-omejena, če je vrednost k maksimalna gledano po vseh pogojih.*

Značilnost omejenosti porajanja števila žetonov v posameznem pogoju je pogoj za možnost stabilne (končne) realizacije modeliranega sistema.

4.5.3 Konservativnost v Petrijevih mrežah

V mnogih primerih uporabe Petrijevih mrež za namene modeliranja nam pogoji predstavljajo vsebinske resurse, ki morajo biti razpoložljivi za izvedbo akcij. Takšnih resursov skozi dinamiko mreže ne moremo niti uničiti, niti povečevati njihovega števila. Zanje velja značilnost ohranjanja ali *konservacije* v smislu njihovega števila. Slednje lahko zapišemo tudi formalno.

Definicija 9 *Petrijeva mreža C=(P,T,I,O) z začetno označitvijo o je striktno konservativna, če za vse o'$\in R(C,o)$ velja relacija $\sum_{p_i \in P} o'(p_i) = \sum_{p_i \in P} o(p_i)$.*

Relacija v definiciji zahteva, da se vsota žetonov skozi dinamiko mreže ne spreminja. Striktna konservativnost v mreži zahteva, da je $\forall j, j = 1, ..., n, n =$

$|T|, |I(t_j)| = |O(t_j)|$. Povedano drugače mora za vsako akcijo veljati, da je število vhodnih povezav enako številu izhodnih povezav.

V večini modelov realnih sistemov striktna konservativnost ni potrebna. Nekateri pogoji npr. lahko predstavljajo števce, v katerih se število žetonov skozi čas spreminja. V takšnih primerih skušamo s primerno obravnavo eliminirati pogoje, v katerih imamo motečo dinamiko in ohraniti pod drobnogledom samo tiste pogoje, v katerih želimo skozi čas imeti vsotno gledano konstantno število žetonov. Zapišimo to formalno.

Definicija 10 *Petrijeva mreža C=(P,T,I,O) z začetno označitvijo o je konservativna glede na utežni vektor* $w=(w_1, w_2, ..., w_n), n = |P|, w_i \in \{0, 1\}$, *če za vse* $o' \in R(C, o)$ *velja relacija* $\sum_{p_i \in P} w_i \cdot o'(p_i) = \sum_{p_i \in P} w_i \cdot o(p_i)$.

Petrijeva mreža je tako lahko striktno konservativna le ob upoštevanju vektorja uteži $w = (1, 1, ..., 1)$, poljubna mreža pa venomer konservativna z vektorjem uteži $w = (0, 0, ..., 0)$. Zaradi slednjega pogoj za konservativnost zaostrimo in sicer zahtevamo, da je vektor w neničeln ($\exists w_i = 1$).

4.6 Zgledi modeliranja s področja računalniških omrežij

V pričujočem razdelku si bomo ogledali še nekaj specifičnejših zgledov modeliranja s Petrijevimi mrežami na področju računalniških omrežij.

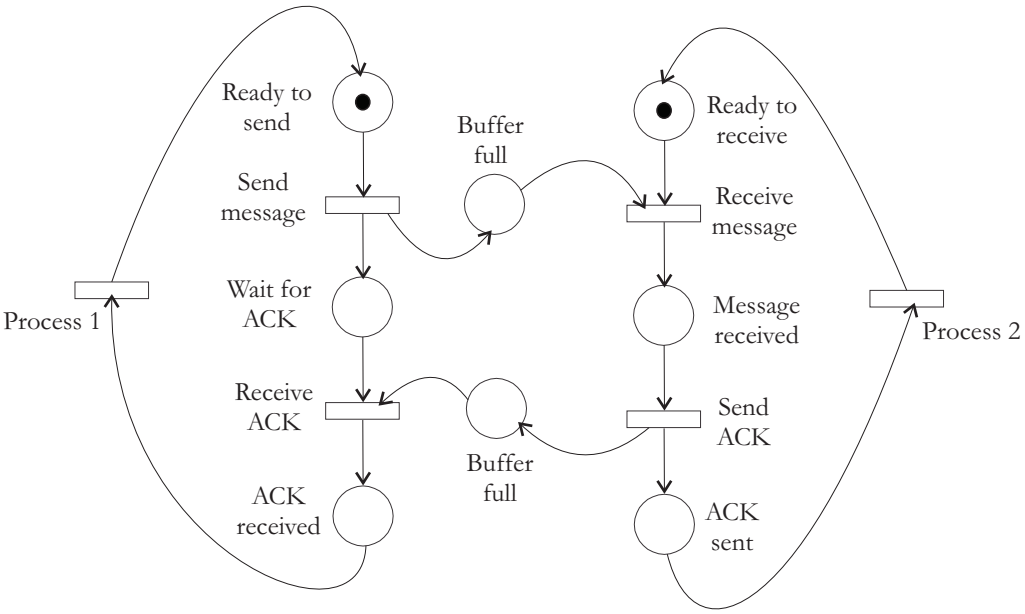
4.6.1 Poenostavljen model protokola med oddajnikom in sprejemnikom

Predpostavimo, da imamo opravka z enostavnim primerom protokola med oddajnikom in sprejemnikom. Prvi sporočila oddaja in počaka na potrditev vsake oddaje, drugi pa sporočila prejema in generira potrditve vseh prejemov. Model tovrstnega sistema je predstavljen na sliki 4.24, povzeti po [6].

V modelu bodimo pozorni na pogoja Bufferfull, saj je iz slike ob podani začetni označitvi razvidno, da sta varna. Enako velja tudi za vse ostale pogoje.

4.6.2 Model nedeterminističnega čakalnega procesa

Predpostavimo, da imamo opravka s sistemom pošiljanja sporočil, pri čemer se odpošiljanje izvede na osnovi nedeterministične izbire akcij t_{r1} (pošiljanje naslovniku 1), t_{r2} (pošiljanje naslovniku 2) ali t_{out} (sporočilo se pošlje v ponovno oddajo, ker predhodno poslanega sporočila ni potrdil nobeden od naslovnikov - simbolična predstavitev stanja TimeOut). Model tovrstnega sistema je predstavljen na sliki 4.25 povzeti po [6].



Slika 4.24: Poenostavljen model protokola med oddajnikom in sprejemnikom povzet po [6].

4.7 Razširitve Petrijevih mrež

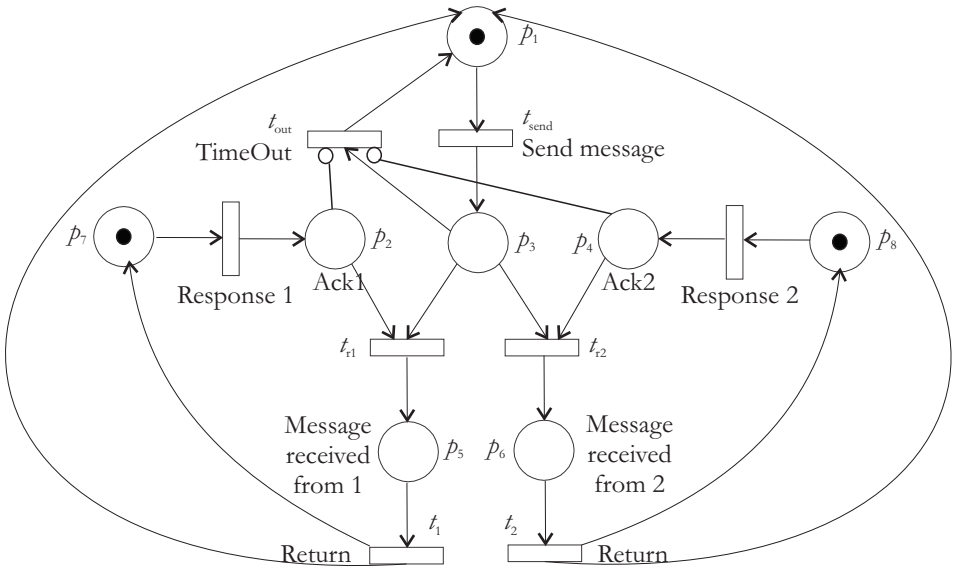
V predhodnjih razdelkih smo si ogledali Petrijeve mreže, kot jih je definiral njihov avtor C. A. Petri. Pri tem smo se v zgledu modela IF stavka in nekaterih kasnejših zgledih že seznanili s konstruktom *inhibirne povezave*, ki ne sodi v osnovno definicijo Petrijevih mrež. Ostale razširjene konstrukte navajamo v naslednjih razdelkih.

4.7.1 Posebni gradniki v Petrijevih mrežah

V razširitvah Petrijevih mrež pogosto zasledimo naslednji vsebinski razširjavi:

- stohastično pogojenost vejanja ali izvedbe akcije, s čimer pridemo do stohastičnih Petrijevih mrež,
- mehkost (angl. *fuzziness*) izvedbe akcije ali izpolnjenosti pogoja, s čimer pridemo do mehkih Petrijevih mrež.

Zaradi splošnosti našega dela omenjenih razširitev ne bomo obravnavali. Poseben primer Petrijevih mrež so *barvne Petrijeve mreže* (angl. *coloured Petri nets*), ki pa jih bomo obravnavali v naslednjem poglavju.



Slika 4.25: Model nedeterminističnega čakalnega sistema povzet po [6].

4.7.2 časovne Petrijeve mreže

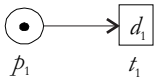
V časovnih Petrijevih mrežah vpeljemo *čas trajanja* posamezne akcije. Njihovo definicijo po [7] zapišemo na naslednji način:

Definicija 11 *šestorček $PN = (P, T, I, O, o(t_0), D)$ imenujemo razširjena časovna Petrijeva mreža, kjer P predstavlja končno množico pogojev, T končno množico akcij, I vhodno matriko in O izhodno matriko, pri čemer obe matriki glasita na akcije. $o(t_0)$ predstavlja začetno porazdelitev žetonov po indeksirani množici pogojev, D pa vektor nenegativnih števil vključujoč ničlo, ki posameznim akcijam določajo časovno trajanje.*

Trajanje posamezne akcije si interpretiramo na sledeči način. Akcija t_j s trajanjem d_j časovnih enot se začne izvajati, ko so izpolnjeni vsi pogoji, iz katerih vodijo vhodne povezave v opazovano akcijo t_j. Pogoji so (in morajo biti) izpolnjeni vse do konca trajanja akcije. šele po d_j časovnih enotah se njihova izpolnjenost razveljavi - žetoni se hipno prenesejo na izhodno stran opazovane akcije. Povedano drugače, žetoni v vhodnih pogojih ostanejo na svojih mestih d_j časovnih enot, šele nato pa se prenesejo preko akcije t_j v izhodne pogoje na nam že znani način. Seveda to ne pomeni, da se število žetonov skozi čas ohranja.

Na sliki 4.26 je prikazan enostaven primer Petrijeve mreže z akcijo t_1 s predvidenim trajanjem d_1 urinih period. če žeton v času t_0 prispe v pogoj p_1, se bo iz pogoja preko akcije t_1 hipno preselil v časovni točki $t_0 + d_1$. Seveda bo to tega prišlo, če v intervalu $]t_0 + d[$ omenjenega žetona ne bo uporabila (prevzela)

kaka druga akcija, ki ji omenjeni pogoj tudi predstavlja potreben vhod. Če bi takšna konkurenčnost ali kompeticija obstajala, uspe z odvzemom žetona tista akcija, ki se preje izvede. če bi bilo torej proženje več akcij odvisno le od enega pogoja (žetona v njem), bi se izvedla tako le ena in sicer tista, z najkrajšim časom izvedbe.

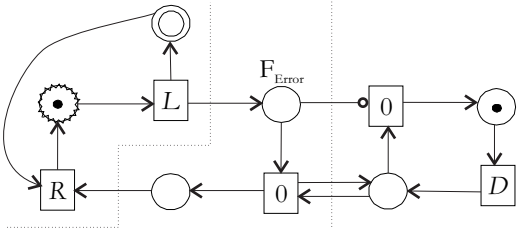


Slika 4.26: Graf Petrijeve mreže s časovnim trajanjem akcije t_1.

Oglejmo si zgled uporabe časovne Petrijeve mreže na primeru servisiranja popravljivega računalniškega sistema.

Zgled 6 *Predvideni čas do odpovedi sistema je L, čas servisiranja pa R časovnih period. Kontrolor periodično (s periodo D) kontrolira sistem in če je sistem v odpovedi, pokliče serviserja.*

Rešitev: Petrijeva mreža opisanega sistema je predstavljena na sliki 4.27, pri čemer levi del slike predstavlja sistem, osrednji del delovanje kontrolorja, desni del slike pa ciklus odsotnosti kontrolorja. Čas zadrževanja kontrolorja v osrednjem delu je hipen. Pri tem pogoj z nazobčanim robom naznačuje delujoče stanje celotnega sistema ($System_{ON}$), pogoj ponazorjen z dvojnim krogom pa začasno nedelujoče stanje celotnega sistema ($System_{OFF}$). Sistem v tem primeru ne vrši predvidene funkcije.



Slika 4.27: Graf Petrijeve mreže za periodično diagnostiko popravljivega računalniškega sistema.

Poglavje 5

Barvne Petrijeve mreže

5.1 Uvod

Barvne Petrijeve mreže (angl. *coloured Petri nets*) so grafično orientiran jezik, ki ga uporabljamo za potrebe načrtovanja, specificiranja, simuliranja in verifikacije dinamičnih sistemov [8], [9]. Uporaben je predvsem na področjih komunikacij, sinhronizacij procesov in v sistemih, kjer prihaja do deljenja resursov. Iz tega vidika so barvne Petrijeve mreže uporabne tudi na področju računalniških komunikacij, kjer se srečujemo z zgoraj naštetimi problemi. Uporabljamo jih predvsem pri snovanju in verifikaciji pravilnosti delovanja komunikacijskih protokolov.

Z razliko od običajnih Petrijevih mrež, se v barvnih Petrijevih mrežah spremeni narava žetona. Slednji dobi pomen *sestavljene podatkovne strukture*, kar pomeni, da takšen žeton omogoča prenos vrednostno inicializiranih spremenljivk (različnih podatkovnih vrednosti predhodno definiranega tipa) po Petrijevi mreži.

Doslej smo se v našem delu usmerjali predvsem na modeliranje prometa med sistemi (angl. *traffic modeling*), v pričujočem poglavju pa se bomo usmerili na zglede modeliranja računalniških komunikacijskih protokolov, ki tovrstni promet omogočajo (angl. *protocol modeling*). Slednji predstavljajo osnovo za kakršnokoli komunikacijo med sistemi v omrežju. Oglejmo si splošno definicijo podatkovnega protokola povzeto po [10].

Definicija 12 *Računalniški komunikacijski protokol definira format in pravila za izmenjavo podatkov preko računalniškega omrežja.*

Po OSI/ISO modelu poznamo 7 komunikacijskih slojev in sicer

- fizični sloj,
- podatkovni sloj,
- mrežni sloj,

- transportni sloj,
- sejni sloj,
- presentacijski sloj,
- aplikacijski sloj,

na vsakem od njih pa uporabljamo različne protokole. V tem smislu se protokoli nalagajo v vertikalni sklad (npr. HTTP protokol sloni na TCP/IP protokolu). V nadaljevanju poglavja si bomo ogledali dva zgleda modeliranja protokolov na podatkovnem sloju. Osnovne naloge omenjenih protokolov so po [10] sledeče:

- na oddajni in sprejemni strani predstavljajo vmesnik med fizičnim in mrežnim slojem,
- skrbijo za korekcijo napak (okvar ali izgub paketov), do katerih pride med prenosom paketov,
- skrbijo za regulacijo in sinhronizacijo pretoka v smislu hitrosti in ostalih zmogljivostnih karakteristik (npr. dolžin čakalnih vrst ali velikosti vmesnikov), ki jih imajo naprave na prenosni poti.

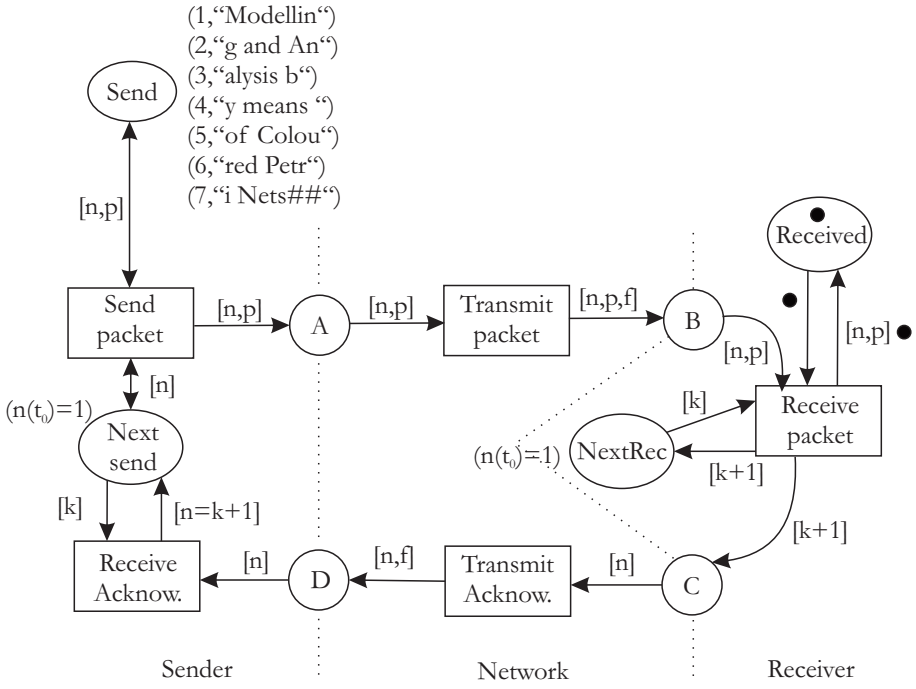
V večini primerov je nižji del podatkovnega sloja realiziran v strojni opremi, zgornji del pa v obliki samostojnega gonilnika (angl. *driver*), ali pa kot del operacijskega sistema.

5.2 Model enosmernega oddajno sprejemnega protokola z neidealno prenosno potjo

Za lažje razumevanje dinamike v tovrstnih Petrijevih mrežah si oglejmo zgled na sliki 5.1 povzet po viru [9]. Predstavlja model enostavnega komunikacijskega protokola za prenos paketov v nezanesljivem omrežju, pri čemer je na levem delu slike predstavljen oddajnik, v osrednjem neidealno izgubno omrežje, v desnem pa sprejemnik. Elipse in krogi predstavljajo *pogoje*, četverokotniki pa *akcije*. Na povezavah so v oklepajih ponazorjeni tipi podatkovnih struktur, ki jim je dovoljena pot po posamezni povezavi. Tipi se lahko med posameznimi povezavami razlikujejo.

Pot pojmom neidalne prenosne poti smatramo pot, na kateri se zaradi vplivov šuma (angl. *noisy channel*) ali poplavne izgube paketov (angl. *flooding*) na poti paketi spremenijo - okvarijo (angl. *corrupted, damaged*) ali izgubijo (angl. *lost completely*) [10]. Za reševanje problema okvarjenih paketov uporabljamo metodi detekcije in korekcije. Prva je časovno in prostorsko manj potratna, a zna le detektirati napake v paketu, ne zna pa jih odpraviti. Druga metoda je časovno in prostorsko bolj potratna (večje število logičnih operacij in večje število redundantnih bitov), a zna pakete tudi popravljati. Zaradi cene metod in pa relativno visoke zanesljivosti prenosnih medijev v večini primerov uporabljamo prvo metodo, ki ob porajanju napak vodi v ponovno pošiljanje paketov.

V tem primeru računamo tudi na dejstvo, da je večina napak na prenosnem mediju minljivih (angl. *transient*), s čimer je eventuelno ponovno pošiljanje (angl. *retransmit*) hipotetično uspešno.



Slika 5.1: Graf barvne Petrijeve mreže za modeliranje enostavnega komunikacijskega protokola med sprejemnikom in oddajnikom s potrjevanjem [9].

Aktivnost dinamike na sliki 5.1 je pogojena s sedmimi začetnimi žetoni v pogoju *Send*, pri čemer je vsak od njih formiran v obliki sporočila (paketa). Posamezni žeton je podatkovno definiran kot par [n,p], pri čemer n predstavlja indeks paketa, p pa njegovo podatkovno vsebino (niz sedmih znakov). Pogoja *NextSend* in *NextRec* sta na začetku inicializirana z žetonoma formata [n], pri čemer je $n = 1$. Pomen obeh pogojev je interpretiran z indeksom paketa (števcem), ki naj bo naslednji oddan ali prejet. Pogoj *Received* predstavlja končno odlagališče uspešno prejetih paketov, pogoji A, B, C in D pa vmesnike med omrežjem in oddajno ter sprejemno stranjo. Slednji na začetku ne vsebujejo žetonov.

Pod drobnogled vzemimo akcijo *SendPacket*. Slednja se lahko sproži pod vplivom pogojev *Nextsend* z žetonom tipa [n] in *Send* z žetonom tipa [n,p], pri čemer se mora vrednost indeksa n v obeh žetonih ujemati. Ker je *Nextsend* na začetku inicializiran z vrednostjo $n = 1$, bo tako na začetku dinamike na izhodni pogoj A (vmesnik pred omrežjem) akcije *SendPacket* poslan žeton [1,$p =$ "*Modellin*"]. Na tem mestu poudarimo, da se zaradi dvosmernih povezav

oba žetona vrneta tudi na ustrezna vhodna pogoja opazovane akcije, kar predstavlja zaščito pred izgubo paketa v omrežju. Šele potrditev prejema paketa s sprejemne strani bo povzročila spremembo pogojev *Nextsend* in *Send*, iz česar odseva pesimistična naravnanost protokola, saj bomo omenjeni paket pošiljali vse dotlej, dokler ne dobimo potrditve prejema žetona (paketa) s strani sprejemnika. Žeton [n,p] v akciji *Transmitpacket* pridobi dodatni podatkovni atribut f, ki ponazarja pravilnost ($f = true$), ali nepravilnost pri prenosu paketa ($f = false$). Žeton se posreduje v vmesnik B samo če je $f = true$. Inicializacija vrednosti f se izvede v akciji *Transmitpacket*, pri čemer v splošnem lahko predpostavljamo, da se vrednost izbira naključno, ali v skladu s statistikami, ki smo jih predhodno zbrali v zvezi z uspešnostjo pravilnega prenosa paketa preko nezanesljivega omrežja. V primeru izgube paketa (žetona), le ta ne pride do akcije *Transmitpacket*.

S prihodom žetona v B postane omogočena akcija *Receivepacket* pod pogojem, da se vrednosti k iz pogoja *NextRec* in n iz matičnega žetona (paketa) ujemata. V tem primeru se paket [n,p] odloži v pogoj *Received*, inkrementira indeks novega pričakovanega paketa v *NextRec* ($k+1$) in pošlje potrditev k-tega paketa v obliki indeksa $k + 1$ (indeks novega pričakovanega paketa) proti vmesniku C. Če se vrednosti k iz pogoja *NextRec* in n iz matičnega žetona (paketa) ne ujemata, matični žeton (paket) obtiči v vmesniku B.

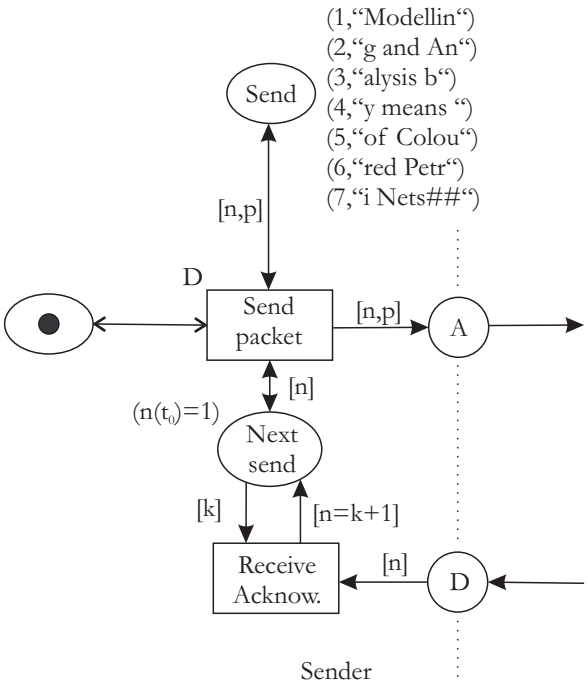
Potrditveni paket [n] nato potuje preko akcije *TransmitAcknow*, ki po vzoru *Transmitpacket* doda atribut f, s katero okarakterizira pravilnost prenosa potrditvenega paketa. Tako v vmesnik D pridejo le tiste potrditve [$n = k + 1$], pri katerih je $f = true$. Akcija *ReceiveAcknow* v pogoj *Nextsend* prenese indeks naslednjega žetona (paketa), ki naj se začne prenašati, odstrani pa stari indeks, s čimer se stari paket ne oddaja več. V celotni mreži tako krožijo v žetonih naslednji tipi spremenljivk

- p: STRING,
- n, k: INTEGER,
- f: BOOLEAN.

Žeton, ki kroži med akcijo *Receivepacket* in pogojem *Received* nima podatkovne vsebine.

Zaradi preglednosti slike ni dodan segment, ki odstrani matični žeton (paket) iz pogoja *Send*, istočasno pa v mrežo nismo vpletli časovnega trajanja akcij, s čimer bi vsaj v segmentu sekvence odpošiljanja paketa $[n, p]$ lahko razbremenili omrežje z vidika prometa. Omenjena izboljšava je parcialno predstavljena na sliki 5.2, pri čemer smo dodali časovnost trajanja D akciji *SendPacket*. Smiselno bi bilo, da je časovna vrednost D trajanja akcije *SendPacket* inicializirana najmanj na pričakovani čas od oddaje podatkovnega paketa do sprejema potrditvenega paketa s strani sprejemnika.

Glede na to, da ne oddajamo novega paketa, dokler ne dobimo potrditve starega, bi lahko protokol poenostavili z indeksiranjem paketov s kodnim naborom $\{0, 1\}$, pri čemer bi se kodi s časom izmenično spreminjali. Poenostavitev bi bila v optimizaciji kompaktnosti kode indeksa n.



Slika 5.2: Dodajanje časa D med posameznimi odpošiljanji paketa.

Še enkrat poudarimo, da lahko pride tako do izgube podatkovnih paketov, kot tudi do izgube kontrolnih potrditvenih (ACK) paketov. Predhodno navedene strategije so realizirane v protokolih z značilnostmi ARQ (angl. *automatic repeat request*) in PAR (angl. *positive acknowledgement with retransmission*).

5.3 Protokol drsečega okna z neidealno prenosno potjo

V prejšnjem zgledu smo ločevali med oddajnikom in sprejemnikom, pa tudi fizični poti paketov smo ponazorili ločeno. Tovrstni prenos imenujemo za enosmerni prenos (angl. *half duplex*). V pričujočem razdelku bomo predpostavili, da je možno pakete pošiljati v obe smeri istočasno (angl. *full duplex*). S tem pride do prepletanja podatkovnih in kontrolnih dvosmernih prenosov po istem mediju. Pri dvosmerni komunikaciji morata sogovornika imeti tako sposobnost sprejemanja, kot tudi sposobnost oddajanja. V pričujočem razdelku bomo dvosmernost prenosa prikazali na okrnjenem zgledu protokola drsečega okna (angl. *sliding window protocol*).

Ena od tehnik, ki se uporablja pri dvosmernem prenosu, je tehnika dodajanja potrditev v podatkovne pakete (angl. *piggybacking*). Osnovna ideja teh-

nike je v tem, da potrjevalni paketi ne potujejo ločeno od podatkovnih, temveč sprejemnik paketa potrditev izvede na ta način, da naslednjemu podatkovnemu paketu, ki ga pošilja proti pošiljatelju, doda tudi potrditev (ACKnowledge) predhodno prejetega paketa. S tem dobimo optimalnejši izkoristek razpoložljive pasovne širine, saj se odposlani podatkovni paket skrajša za naslov prejemnika in redundantne bite samostojnega ACK paketa. Seveda pa se na tem mestu poraja problem po kolikšnem času bo prišlo do izdaje podatkovnega paketa s strani naslovnika, ki je paket prejel od pošiljatelja. Lahko da bo tovrstni podatkovni paket prispel z dodano ACK vsebino prepozno in bo izvorna stran zaradi vgrajenega časovnega zamika potrditve že pred prejemom podatkovni paket proti sprejemniku poslala ponovno. Glede na povedano v večini primerov sprejemnik pozna časovni zamik čakanja oddajnika na potrditev. V primeru, da sprejemnik nima pripravljene podatkovne vsebine, mora ACK poslati proti pošiljatelju pred iztekom vnaprej določenega časa v samostojni paketni obliki.

Druga karakteristika protokola drsečega okna je možnost odpošiljanja večjega števila paketov proti naslovniku (npr. m paketov), pri čemer se na oddajni strani vodi lista (okno) m oddanih paketov, na potrditev katerih čaka oddajnik. Vsak potrjeni paket s strani prejemnika se odstrani iz okna, odda pa se lahko nov paket, z oddajo pa se uvrsti tudi v okno. V tem smislu okno oddajnika vodi evidenco paketov, za katere mora od sprejemnika še dobiti potrditev. Omenjena evidenca o stanju okna mora biti izvedena tako na eni, kot tudi na drugi strani dvosmerne komunikacije, saj je funkcionalnost obeh točk dvojna; vsaka točka ima tako funkcijo oddajnika, kot tudi sprejemnika.

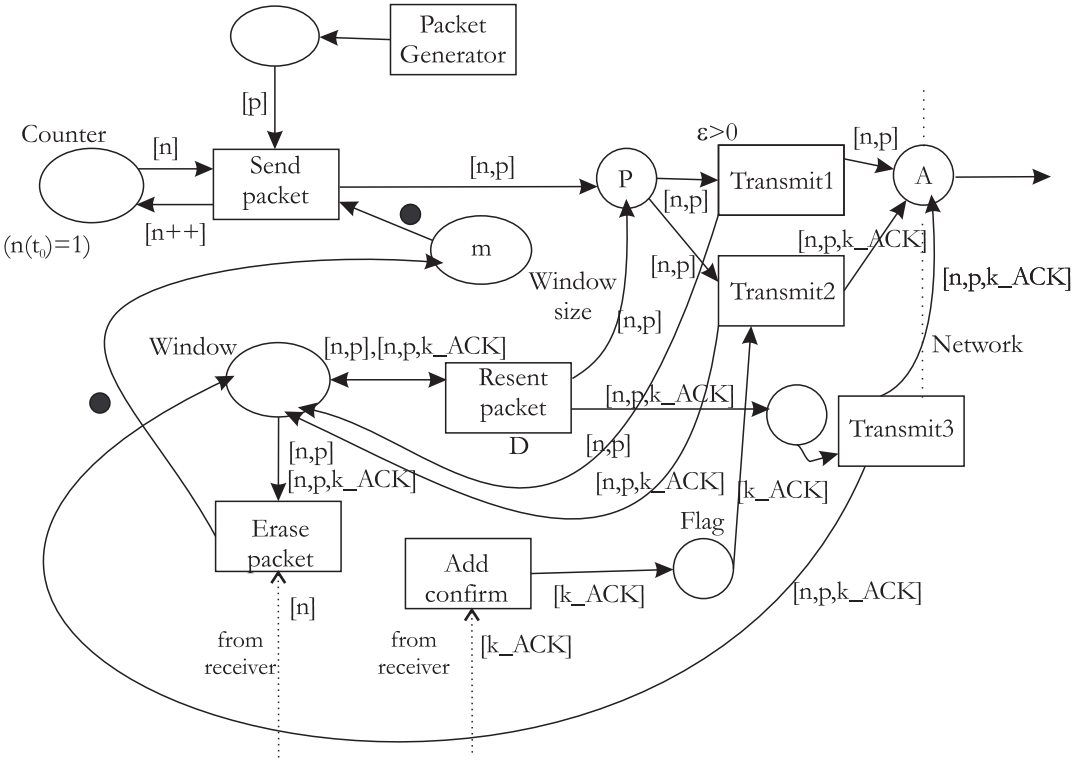
Na sliki 5.3 je predstavljen graf Petrijeve mreže, ki ponazarja oddajni del enega od sogovornikov. Akcija *PacketGenerator* generira prvih m paketov, v nadaljevanju pa še posamezne pakete, v odvisnosti od števila potrditev, ki v nadaljevanju prihajajo v pogoj *WindowSize*. Pri tem pogoj *Counter* skrbi za inkrementacijo indeksa novoustvarjenega paketa. Paketi se poleg pošiljanja v omrežje skladiščijo tudi v pogoju *Window*, ki vrši funkcijo drsečega okna. Vsak paket, ki prispe v ta pogoj, se preko akcije *ResentPacket* odpošlje proti omrežju vsakih D urinih period, če seveda predhodno iz okna ni odstranjen preko akcije *ErasePacket*. Pogoj za to je prispetje žetona $[n]$ s strani sprejemnega dela prvega sogovornika. Dokončna oddaja se vrši preko treh različnih akcij. Njihove pomenske značilnosti so naslednje:

- akcija *Transmit1* oddaja novoustvarjene ali predhodno že poslane pakete, ki jim je potekel čas prejema potrditve formata $[n, p]$ brez dodatnega prenosnega bremena potrditve prejetega paketa na sprejemni strani; da bi se paket iz pogoja P odločil za izbiro akcije *Transmit1*, ne pa *Transmit2*, ne sme biti vzpostavljen pogoj *Flag*, kar je doseženo s časovnostjo akcije $\epsilon > 0$; oddani paket je formata $[n, p]$;

- akcija *Transmit2* oddaja novoustvarjene ali predhodno že poslane pakete, ki jim je potekel čas prejema potrditve formata $[n, p]$; doda se jim čakajoča potrditev prejetega paketa, tako da je končni oddajni format $[n, p, k\_ACK]$;

- akcija *Transmit3* oddaja predhodno že oddane pakete formata $[n, p, k\_ACK]$, za katere potrditev prejema ni prispela pravočasno; slednjim ne dodaja eventuelnih novih potrditev, ki čakajo v pogoju *Flag*;

- vse akcije poslane pakete odložijo tudi v pogoj *Window*;



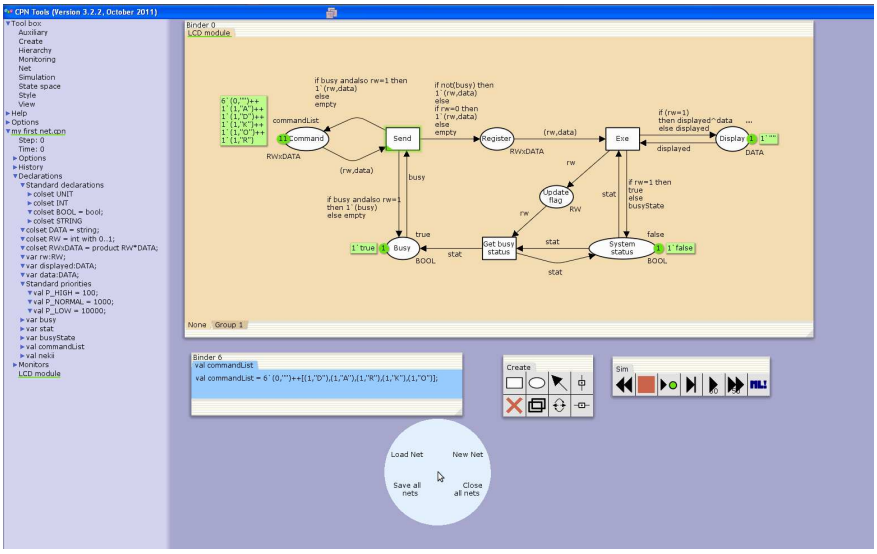
Slika 5.3: Oddajni del prvega sogovornika v komunikaciji.

Poleg akcij *Transmit1*, *Transmit2* in *Transmit3* bi morali na sliko 5.3 uvrstiti še časovno opredeljeno akcijo *Transmit4*, ki bi zagotavljala odvod paketov tipa $[i, k\_ACK]$ po določenem času, če za prenos ni bil pripravljen noben podatkovni paket ali njegova retransmisija preko akcije *Transmit2*. Omenjene akcije na sliko zaradi preglednosti nismo uvrstili.

Na podoben način bi lahko realizirali tudi sprejemno stran posameznega sogovornika v sistemu. Obsežnejši zgled modela protokola drsečega okna (angl. *sliding window*) je s pomočjo grafov Petrijevih mrež opisan in prikazan v delu [11].

5.4 Programsko orodje CPN Tools za delo z barvnimi Petrijevimi mrežami

Za lažje delo z barvnimi Petrijevimi mrežami je razvitih kar nekaj programskih orodij. Eno od popularnejših nekomercialnih in prosto dosegljivih je orodje *CPN Tools* [12]. Na sliki 5.4 je predstavljen uporabniški vmesnik z naloženim modelom. Orodje omogoča osnovne funkcije gradnje modela (dodajanje pogojev, akcij, povezav, itd.), modularnost izgradnje modela, upravljanje s podatkovnimi tipi žetonov in samodejno preverja sintaksno pravilnost modela. Posamezni model se shrani v datoteko s končnico *.cpn*, ki temelji na XML datotečni strukturi.



Slika 5.4: Uporabniški vmesnik orodja *CPN Tools* [11].

Orodje ponuja tudi izvajanje simulacij na osnovi zasnovanega modela. Slednje se izvajajo grafično. Tako lahko skozi čas spremljamo število žetonov v posameznih pogojih. Osnovne funkcije za izvajanje simulacij delimo na *interaktivne* in *samodejne*. Pod interaktivnimi imamo v mislih tiste, kjer se del simulacijske decizije prepušča uporabniku (npr. ročno izvajanje akcij, ročno izbiranje konkurenčnih ali splošneje omogočenih akcij), pod samodejnimi pa samodejno (avtomatizirano) izvajanje vključno z reševanjem nedeterministične izbire vejanj, reševanja konkurenčnosti, izvajanje simulacij za vnaprej predvideno število simulacijskih korakov ali realen čas, ali do izpolnitve vnaprej podanih pogojev (npr. konkretne označitve), itd.

Pri proženju samodejne simulacije brez definicije konca, ki bo tekla vse do dosega končne označitve, je pomembno, da se zavedamo, da ni nujno, da se bo simulacijski tok ustavil. Do tega lahko pride bodisi, ker v sistemu ni končne označitve, ali pa zaradi tega, ker se je sistem vsled nedeterminističnim vejanjem

slednji izognil. Tovrstni dogodek nas že opozarja, da obstaja možnost preslabe definicije zasnove rešitve problema (modela).

Orodje za uspešnejše razhroščevanje omogoča vpeljavo *nadzornikov*, ki imajo vnaprej predvidene nadzorne funkcije. Slednje definirajo pogoje pod katerimi pride do začasne (k interakciji je povabljen uporabnik), ali dokončne ustavitve simulacije. Podpira tudi analizo prostora dosegljivih stanj. Rezultate analize orodje predstavi z usmerjenim grafom, v katerem kot vozlišča nastopajo označitve (stanja sistema). Slednje smo v poglavju o Petrijevih mrežah predstavili z *drevesom dosegljivih stanj*. Analiza prostora stanj nam poleg drevesa dosegljivih stanj predstavi še naslednje pomembne značilnosti modeliranega sistema:

- velikost prostora stanj,
- obstoj k-omejenosti z isto barvnimi žetoni v posameznih pogojih,
- obstoj domače označitve, ki je dosegljiva iz vseh preostalih označitev (dobrodošla značilnost, ki ponazarja, da se je sistem po opravljeni nalogi sposoben vrniti v neko začetno stanje),
- pogostosti proženja posameznih akcij, itd.

Dodatne primere modelov barvnih Petrijevih s področja računalniških komunikacij najdemo v delu [13].

Poglavje 6

Ključne metrike in orodja za ocenjevanje zmogljivosti računalniških omrežij

6.1 Uvod

V pričujočem poglavju si bomo ogledali nekaj ključnih metrik za ocenjevanje računalniških omrežij, na katere moramo biti pozorni tako pri njihovem snovanju, kot tudi pri samem spremljanju prometa. Ogledali si bomo pojme *latence*, *zgostitve prometa* in *strukturalne neuravnoteženosti resursov*, istočasno pa tudi navedli vrste programskih in strojnih orodij, ki nam olajšajo delo pri analizi delovanja omrežij.

6.2 Latenca v računalniških omrežjih

Pojem *latence* v splošnem definiramo kot *časovni zamik* med trenutkom ko se je neko dejanje sprožilo in trenutkom, ko nastopijo opazni učinki tega dejanja. Med samim trajanjem časa latence so učinki dejanja latentni, potencialni, oziroma jih še ni mogoče zaznati. Z uporabniškega vidika so tipični primeri časa latence čas nalaganja spletne strani, čas potovanja elektronske pošte od pošiljatelja do naslovnika itd. Celotni čas latence sestavljajo čas prenosa, čas obdelave in čas, ki ga zahteva ali paket porabi za čakanje na poti ali pred fazo obdelave. V primeru računalniških omrežij je osnovna enota latence čas v mili sekundah (ms). Latenca je pogojena z dolžino poti med oddajnikom in sprejemnikom, vrste prenosnega medija, vrsto in zanesljivostjo delovanja protokola (korekcijske ali detekcijske metodologije) in zastoji paketov zaradi čakanja v čakalnih vrstah.

Pojem *latence* v računalniškem omrežju lahko definiramo na sledeče različne načine:

- z vidika *posameznega paketa* je latenca čas, ki ga potrebuje paket za pot od izvora (pošiljatelja) do ponora (naslovnika);

- z vidika *posamezne zahteve* je latenca čas, ki ga potrebuje zahteva za pot od izvora do ponora (naslovnika); ni nujno, da je zahteva sestavljena le iz enega paketa;

- z vidika *posamezne zahteve in odgovora nanjo*, je latenca sestavljena iz časa, ki ga potrebuje zahteva za pot od izvora do ponora, časa za procesiranje odgovora na strani ponora in časa, ki ga odgovor potrebuje za pot od ponora nazaj do izvora (angl. *round trip time*).

V pričujočem delu se bomo naslonili na prvo od treh definicij. Latenca je neposredno odvisna od pasovnih širin prenosnih medijev na komunikacijski poti, hitrosti delovanja aktivnih omrežnih komponent, količine prometa, fizične razdalje med ponorom in izvorom, ter eventuelnih motenj (npr. vremenskih) na omrežju. Glede na podani izvor in ponor se latenca običajno skozi čas spreminja. Za določanje latence in njena merjenja obstajajo javno dostopni standardi (npr. RFC1242, RFC2544, itd.).

6.2.1 Natančnejša definicija latence proizvajalca Siemens

Kanadski proizvajalec omrežnih produktov RuggedCom Inc., ki ga je kupilo podjetje Siemens, vpelje naslednjo matematično notacijo sestavnih delov latence v omrežju [14]:

- *shranjevalna in posredovalna latenca* L_{SF} (angl. *store and forward latency*) izvira iz obdelave na aktivni napravi; takšna naprava del latence doprinese s časom shranjevanja prejetih podatkov v pomnilnik, dokler celotni paket ni sprejet; drugi del časa k latenci doprinese prenašanje paketa na ustrezni izhod naprave; oba doprinosa zapišemo s spremenljivko L_{SF} z izrazom

$$L_{SF} = \frac{FS}{BR}, \tag{6.1}$$

  pri čemer je FS velikost prenašanega paketa v bitih in BR bitna hitrost prenosnega medija (v bitih na sekundo); pri paketu velikosti 1500 byte-ov (največja efektivna velikost Ethernet paketa) in prenosnem mediju 100 Mbps je tako latenca 120 μs, pri paketu velikosti 64 byte-ov (najmanjša efektivna velikost Ethernet paketa) in prenosnem mediju 1 Gbps pa je latenca 0.5 μs;

- *tovarniška latenca aktivne naprave* L_{SW} (angl. *switch fabric latency*) je čas, ki ga k skupni latenci doprinese fizična realizacija logike, ki jo opravlja aktivna naprava; omenjeni proizvajalec za svoje naprave na tem mestu jemlje konstanto 5.2 μs;

- *latenca ožičenja* L_{WL} (angl. *wireline latency*) je čas, ki ga doprinese prenos pa posameznem fizičnem prenosnem mediju, ki povezuje dve aktivni napravi; če je fizični medij npr. optično vlakno, lahko posamezni biti po njem potujejo z dvema tretjinama hitrosti svetlobe ($\frac{2}{3} * 3 * 10^8 m/s$); predvsem pri daljših povezavah ne smemo zanemariti vpliva dolžine prenosnega medija; pri razdalji 1000 km in izbiri optičnega prenosnega medija, bi tako latenco izračunali po izrazu

$$L_{WL} = \frac{1 * 10^6 m}{\frac{2}{3} * 3 * 10^8 m/s} \approx 5ms; \tag{6.2}$$

- *latenca čakanja* L_Q (angl. *queuing latency*) je čas, ki ga paket preživi v čakalnih vrstah pred aktivnimi napravami; čakalne vrste pred posameznimi fizičnimi prenosnimi mediji zanemarimo, ker moram biti sistem ne glede na količino prometa zasnovan tako, da je prepustnost prenosnega medija kos intenzivnosti prihajanja paketov iz aktivne naprave; L_Q zapišemo z izrazom

$$L_Q = NetworkLoad * L_{SF(\max)}, \tag{6.3}$$

  pri čemer je L_Q povprečna latenca iz naslova čakanja, $NetworkLoad$ procentualni delež maksimalnega možnega bremena prometa, $L_{SF(\max)}$ pa L_{SF} največjega možnega paketa (1500 byte-ov), kar pomeni, da je L_Q ocenjena pesimistično.

Glede na štiri različne sestavne tipe skupne latence na komunikacijski poti, bi za komunikacijsko pot s slike 1.1 lahko podali za izračun celotne latence za posamezni paket dolžine FS izraz

$$L_{Total}(paket) = \sum_{i=1}^{n}(L_{SFi} + L_{SWi} + L_{Qi}) + \sum_{i=1}^{n-1} L_{WLi}, \tag{6.4}$$

pri čemer je na sliki 1.1 n aktivnih naprav in $n-1$ fizičnih prenosnih medijev.

Zgled 7 *Predpostavimo, da naš paket dolžine 1500 byte-ov potuje do 7.000 km oddaljene lokacije v ZDA pri 50% obremenjenosti omrežja na celotni liniji. Na svoji poti obišče 10 aktivnih naprav, vsi prenosni mediji pa so optični. Rešitev: Na osnovi izraza 6.4, bi lahko izračunali vrednost izraza*

$$L_{Total}(paket) = \sum_{i=1}^{10}(120\mu s + 5,2\mu s + 60\mu s) + 35ms \approx 35,8ms. \tag{6.5}$$

Do podobnega rezultata, kot smo ga izračunali v pričujočem zgledu, bi prišli, če bi latenco preverili z ukazom `traceroute` ali ukazom `tracert`, ki ju pozna večina operacijskih sistemov in omogočata opis prepotovane poti z ustreznimi

latencami. Iz zgleda je razvidno to, da večina latence (približno 35 ms) izhaja iz prenosnih medijev, le manjši del (približno 1 ms) pa iz aktivnih naprav. Seveda pa temu ob zgostitvah prometa ni tako.

Tipične velikosti latenc (od izvora do ponora) v današnjih računalniških omrežjih se gibljejo od nekaj ms do nekaj 100 ms, če se signal prenaša po prenosnih poteh z nosilnimi mediji kot sta bakrena parica ali optično vlakno. V primeru satelitskih ali brezžičnih povezav, so te latence nekoliko višje. Z vidika uporabnikov so željene latence do 50 ms, saj slednje zadostujejo tudi nemotenemu prenosu signala digitalne televizije.

6.2.2 Natančnejša definicija latence proizvajalca O3b Networks

Proizvajalec O3b Networks ponuja drugačno definicijo latence. Definira jo kot vsoto propagacijske, serializacijske, usmerjevalno - preklopne in latence, ki izhaja iz naslova čakalnih vrst. Več o tem si lahko bralec prebere v viru [15].

6.3 Zgostitve prometa

Pri modeliranju prometa v navideznem (nastajajočem) omrežju ali opazovanju prometa (angl. *monitoring*) v že obstoječem omrežju moramo biti posebno pazljivi na tiste točke v omrežju, v katerih prihaja pogosto ali občasno do zgostitev prometa (angl. *traffic congestion*). Za zgostitev smatramo situacijo, ko pride do čakanja zahtev v čakalnih vrstah pred posameznimi napravami. Z vidika posega v nadgradnjo - izboljšavo modeliranega ali realnega omrežja moramo oceniti naslednje dejavnike:

- časovno pogostost ali frekvenco tovrstnih dogodkov (večja je frekvenca zgostitev, bolj moramo razmišljati o dodajanju resursov),
- velikost ali kritičnost zgostitve prometa z definiranim vplivom na uporabnika (ali in v kolikšni meri uporabnik občuti zgostitev prometa),
- možnost (izvedljivost) ter finančno in časovno potratnost izboljšave, ki se realizira z zvečanjem sistemskih resursov (strežnikov, aktivnih naprav, povezav, itd.) v omrežju,
- identifikacijo vzrokov za zgostitve (vzroki lahko izhajajo iz normalnega naraščajočega prometa, ali zaradi anomalij v omrežju (napadi, nepravilnost delovanja protokolov, itd.)),
- možnost rekonfiguriranja aktivnih naprav, itd.

6.4 Strukturalna neuravnoteženost resursov v omrežju

Pod pojmom strukturalne neuravnoteženosti resursov v omrežju smatramo neustrezno število in razporeditev resursov (aktivnih komponent, povezav, strežnikov, itd.), ki sestavljajo naše omrežje. Manifestira se s porajanjem zgostitev prometa, ki jih povzročajo ozka grla sistema. Običajni ukrepi, ki nas rešijo iz neuravnoteženosti, so uvajanje novih resursov, prestavljanje in prerazporejanje obstoječih resursov, ter njihova nadgradnja. Glede na padajoče cene vpeljave aktivnih komponent je edina resna ovira za doseganje strukturalne uravnovešenosti resursov v omrežju razpoložljivost z dovolj zmogljivimi fizičnimi povezavami.

6.5 Orodja za ocenjevanje zmogljivosti računalniških omrežij

Na koncu drugega poglavja smo navedli dva tipa modelirno simulacijskih programskih okolij in sicer *univerzalne simulacijske jezike* in *aplikacijsko orientirane simulatorje*. Poleg njih poznamo še naslednja orodja, ki se pogosto uporabljajo pri analizi računalniških omrežij:

- Emulatorji - emulatorji omrežij z nastavljivimi parametri; pod parametri imamo v mislih pasovno širino, latenco, izgubo, zgostitve, itd. (angl. *bandwidth*, *latency*, *loss*, *congestion*, *jitter*); koristni so predvsem pri formiranju bremen in analizi odzivnosti sistema nanje;
- mrežni monitorji, generatorji in analizatorji za beleženje, generiranje in analizo prometa;
- generatorji dnevniških datotek (angl. *logging*);
- stresna spletna orodja (angl. *WebServer stress tools*), saj mnogo aplikacij prehaja v spletno obliko;
- uravnoteževalna orodja (angl. *balancing tools*) za uravnoteženje sistemskih resursov (npr. razmerja upload/download, določanja prioritet posameznim protokolom na osnovi koncepta DPI (angl. *deep packet inspection*), itd.

6.6 Vloga generatorjev psevdo naključnih števil

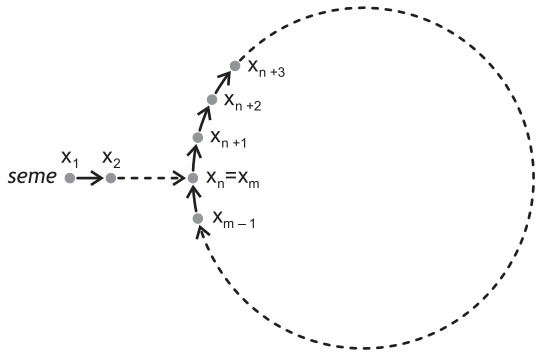
Pri generatorjih prometa moramo biti pozorni na vgrajene *generatorje psevdo naključnih števil*. Slednji generirajo psevdo naključna števila s pomočjo katerih generatorji prometa določajo naključno pogojene medprihodne čase in dolžine paketov [3]. V večini primerov generatorji psevdo naključnih števil temeljijo na iterativnih enačbah, ki jih v splošnem lahko zapišemo z izrazom

$$x_n = f\left(x_{n-1}, x_{n-2}, \ldots, x_0\right), \tag{6.6}$$

pri čemer x_n predstavlja n-to izračunano psevdo naključno število. Primer enostavne tovrstne funkcije je prikazan v izrazu (6.7), kjer je naslednje psevdo naključno število odvisno samo od trenutnega psevdo naključnega števila. Iz izraza je razvidno, da je vrnjeni rezultat venomer celo število med 0 in 15, kar pomeni, da je generator zmožen tvorbe 16 različnih števil in da se začnejo psevdo naključna števila dokaj hitro ponavljati s periodo ponovitev 16. V primeru, da se izračun začne s številom 3 ($x_0 = 3$), je generirano zaporedje psevdo naključnih števil 0, 1, 6, 15, 12, 13, 2, 11, 8, 9, 14, 7, 4, 5, 10, 3, 0, ... Iz zaporedja je razvidno, da se po 16 korakih začnejo števila ponavljati.

$$x_n = (5x_{n-1} + 1) \ \mathbf{mod}\ 16. \tag{6.7}$$

Število x_0, s katerim se začne izračun psevdonaključnega generatorja, se imenuje *seme* (angl. *seed*). Praviloma ima pri generatorju uporabnik možnost določanja semena, a se v večini primerov za seme uporablja vnaprej določena konstantna privzeta vrednost, zato je rezultat naključnega generatorja ob zagonu vedno enak. V nekaterih primerih je za seme privzeta sistemska spremenljivka, ki se zelo hitro spreminja, zato je seme lahko odvisno tudi od časa. Primer takšne spremenljivke je ura realnega časa, in sicer predvsem njene stotinke ali tisočinke sekunde. Na sliki 6.1 je prikazan splošni cikel generatorja psevdo naključnih števil.



Slika 6.1: Cikel generatorja psevdo naključnih števil.

Seme je lahko tudi število, ki ne spada v množico števil, ki sestavljajo cikel. Dober naključni generator mora biti računsko enostaven, da za izračun ne porabimo preveč procesne moči. Perioda ali dolžina cikla generatorja mora biti čimvečja, sicer se začnejo generirana števila prehitro ponavljati. Sosednja števila v zaporedju morajo biti statistično neodvisna in enakomerno porazdeljena. V literaturi lahko najdemo več primerov generatorjev, ki pa so po večini vezani na dolžino besede računalnika. Primeri takšnih generatorjev so podani v izrazih v nadaljevanju.

$$x_n = 7^5 x_{n-1} \ \mathbf{mod}\ \left(2^{31} - 1\right), \tag{6.8}$$

$$x_n = \left(2^{16} + 3\right) x_{n-1} \ \mathbf{mod}\ 2^{31}, \tag{6.9}$$

$$x_n = \left(2^8 + 3\right) x_{n-1} \textbf{ mod } 2^{15}, \tag{6.10}$$

$$x_n = 2^{13} x_{n-1} \textbf{ mod } 2^{35}. \tag{6.11}$$

Poglavje 7

Pridobivanje kvantitativnih vrednosti sistemskih spremenljivk omrežja

7.1 Uvod

V poglavju o splošni teoriji strežbe smo na matematičen način ponazorili relacije vpliva *osnovnih sistemskih spremenljivk*, kot sta intenzivnost prihajanja zahtev λ in intenzivnost strežbe μ, na ostale numerične pokazatelje zmogljivosti računalniškega omrežja (npr. na povprečni čas strežbe in povprečni čas čakanja v čakalni vrsti). Če hočemo zgraditi verodostojen model dinamike opazovanega omrežja, potrebujemo čim natančnejše ocene vrednosti navedenih sistemskih spremenljivk. Do slednjih lahko pridemo na naslednja dva načina:

- *z meritvami*: v primeru, da omrežje že obstaja in imamo v njem možnost izvajati meritve karakteristik prometa, vrednosti osnovnih sistemskih spremenljivk pridobimo s pomočjo izvedbe meritev;
- *s približnimi ocenami*: v primeru, da omrežje še ne obstaja, ali v že obstoječem omrežju ne moremo ali ne smemo opravljati meritev karakteristik prometa in s tem posredno pridobiti ocene sistemskih spremenljivk, se zatečemo k približnim ocenam teh vrednosti; ocene vrednosti osnovnih sistemskih spremenljivk tako pridobimo na osnovi našega poznavanja prometa v podobnih - sorodnih omrežjih;

Termin *meritev omrežnega prometa* (angl. *network traffic measurement*) nakazuje na enkratnost ali občasnost njihovega izvajanja. Meritve tako izvajamo pred eventuelnimi odločitvami o *izgradnji* ali *nadgradnji* omrežja, ciklično pa zaradi pridobivanja informacij o *količini* in *vsebini* omrežnega prometa.

Večina ponudnikov internetnih storitev (angl. *internet service provider* - ISP) ali vzdrževalcev omrežij z vidika zagotavljanja dovoljšnje kvalitete stori-

tev, predpisane v pogodbenem razmerju (angl. *service level agreement* - SLA), tovrstne meritve izvaja neprestano. V tem primeru govorimo o *monitoringu omrežnega prometa* (angl. *network traffic monitoring*).

V računalniških omrežjih običajno tako občasne meritve, kot tudi monitoring vršimo z zajemanjem podatkov v diskretnih časovnih intervalih. Povedano drugače, vrednosti sistemskih spremenljivk *vzorčimo*. *Frekvenca vzorčenja* je odvisna od namena spremljanja dogajanja v omrežju. V primeru enkratnih ali občasnih meritev so frekvence vzorčenja večje, v primeru monitoringa (rednega spremljanja dogajanja v omrežju) pa manjše. Zavedati se moramo, da se *zahteva* za zajem podatkov (zahteva za posameznim vzorcem) s strani sistema vzorčenja pošilja preko omrežja, pa tudi *odgovor* v obliki vzorca prihaja proti sistemu vzorčenja preko omrežja. Vsled temu moramo biti pazljivi, da s preveliko frekvenco vzorčenja pretirano ne obremenimo omrežja. V slednjem primeru govorimo o *invazivnosti vzorčenja* (invazivnosti meritev ali invazivnosti monitoringa), ki nam količino prometa v omrežju povečajo, poveča pa se tudi zasedenost sistemskih resursov aktivnih naprav (npr. procesorja v aktivni napravi) v omrežju. Izkušnje slovenskih operaterjev in vzdrževalcev omrežij kažejo, da je optimalna frekvenca vzorčenja v primeru monitoringa velikostnega razreda nekaj minut. Frekvenca vzorčenja v primeru meritev je odvisna od namena meritev in je običajno večja, v redkejših primerih pa tudi manjša od tiste, ki se uporablja pri monitoringu.

7.2 Vzorčenje kvantitativnih vrednosti opazovanih spremenljivk

Spremenljivke, katerih vrednosti vzorčimo v omrežjih, delimo na naslednje skupine:

- *vzorčenje količine prometa*: količino prometa običajno vzorčimo na aktivnih napravah v diskretnih časovnih intervalih; vrednost vzorčene spremenljivke se ponazarja v obliki števila paketov na časovno enoto, ali še pogosteje v obliki KB (MB) na časovno enoto (npr. sekundo, minuto ali uro);

- *vzorčenje vrste prometa*: v zadnjem času skušajo ponudniki internetnih storitev pridobivati tudi podatke o vrstah prometa, ki poteka preko njihovih omrežij (angl. *deep packet inspection* - DPI); tovrstne podatke pridobivajo predvsem z vidika eventuelnega upočasnjevanja prometa določenih neplačanih storitev in dajanje prioritet vrstam prometa, ki so plačane s strani večjih ponudnikov storitev (npr. Google) ali samih naročnikov; takšno politiko podpira tudi EU, ki je v letu 2015 sprejela zakonodajo, ki tovrstni „internet dveh hitrosti“ legalizira na nivoju EU [16]; s tem se gre v prihodnosti bati za do sedaj veljavno nevtralnost internetnih povezav (angl. *net neutrality*); enostaven primer programa za opazovanje vsebine prometa je `wireshark`;

- *vzorčenje zasedenosti sistemskih resursov*: v tem primeru imamo v mislih vzorčenje zasedenosti resursov na aktivnih napravah (npr. usmerjevalnikih), kjer običajno merimo zasedenost dinamičnega pomnilnika in procesorja, kot osnovnih skalabilnih resursov v aktivnih napravah; meritve zasedenosti pomnilnika nas vodijo do hipotez o (ne)ustreznostih nastavljenih velikosti čakalnih vrst in hipotez o obstoju eventuelnih pomnilniških razpok (angl. *memory leaks*); posledično se v nadaljevanju odločamo o eventuelnem skaliranju pomnilniških resursov, ponastavitev dolžin čakalnih vrst, resetiranju aktivnih naprav in iskanju napak, ki predstavljajo vzroke za pomnilniške razpoke na aktivnih napravah;
- *vzorčenje vrednosti ambientalnih podatkov*: v tem primeru imamo v mislih meritve podatkov o ambientalnih razmerah v prostorih, v katerih se nahaja aktivna oprema; eden od osnovnih merjenih podatkov je ambientalna temperatura, ki naj ne bi presegala neke gornje meje, ki jo določa proizvajalec aktivne opreme; običajno poleg temperature beležimo tudi delovanje ventilatorjev na aktivnih napravah, ki neposredno omogočajo hlajenje posameznih komponent, posredno pa čimbolj enakomerno ambientalno temperaturo;
- *vzorčenje neposrednih performans omrežja*: v tem primeru imamo v mislih pošiljanje paketov na ponorno točko s sprotnim merjenjem časov potovanja paketov; pri tem istočasno ugotovimo, po kateri poti so paketi potovali in ali je ponorna naprava sploh dosegljiva; tipični enostavni računalniški programi za tovrstne meritve, ki so integrirani v operacijski sistem sistema, ki podatke vzorči, so sledeči:
  - `ping`, `traceroute`: identifikacija dosegljivosti ponora in poti paketov;
  - `ping`: meritev latence paketa na poti (časa potovanja paketa);
  - `iperf`: meritev propustnosti poti;
  - `netstat`: meritev napak, ki se porajajo na poti;
- *vzorčenje števila uporabnikov na neki napravi*: v tem primeru vzorčimo podatke na lokalnih aktivnih napravah, kjer se število uporabnikov ne povzpne čez neko predvideno gornjo mejo (npr. vzorčenje na lokalnem stikalu);

7.3 Vplivnostni faktorji na izide meritev

Na prvi pogled z meritvami prihajamo do najmerodajnejših ocen o dogajanju v omrežju. Slednja predpostavka drži, če se kot pripravljalec in izvajalec meritev zavedamo nevarnosti, ki na nas prežijo na tem področju.

7.3.1 Sistemska napaka in šum meritev

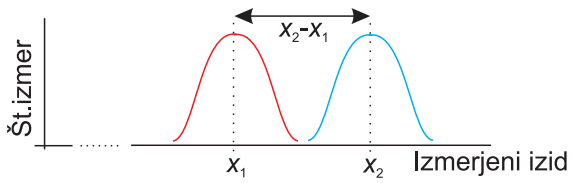
V splošnem se moramo pri vseh vrstah meritev zavedati nevarnosti prisotnosti *sistemske napake* in *šumnosti meritev*. Oba primera lahko razložimo na primeru

meritev, kjer z ročno merilno uro (štoparico) večkrat izmerimo čas teka atleta na 100 metrov.

V primeru, da je naša ura prepočasna, bodo vsi izmerjeni izidi tekov premajhni. V tem primeru govorimo o sistemski napaki, ki se je prikradla v meritev zaradi neustrezne zasnove merilnega postopka (izbire prepočasne ure). Praviloma se takšne napake običajno ne zavedamo, moramo pa v vsaki postavitvi meritve nanjo računati.

Predpostavimo, da delamo meritve vsak dan na začetku treninga, ko je atlet še spočit. V idealnih razmerah bi tako atlet moral doseči vsak dan isti izid. Če predpostavimo tovrstne (sicer nerealne) idealne razmere, bi se rezultati meritev vseeno do neke mere razlikovali, saj bi se v meritev prikradel šum, ki bi izhajal iz naše nenatančnosti ročnega proženja merilne ure. V tem primeru govorimo o prisotnosti šumnosti meritev.

Na sliki 7.1 je grafično ponazorjen primer sistemske napake in šumnosti meritev. Pri tem je na levem delu slike z rdečo barvo ponazorjeno število izmer posameznega izida upoštevajoč sistemsko napako, na desnem delu slike pa z modro barvo število izmerjenih izidov z njihovimi pravimi vrednostmi. Velikost sistemske napake je pogojena z vrednostjo $|x_2 - x_1|$, šumnost meritve pa z standardnim odklonom krivulje obarvano z rdečo.



Slika 7.1: Sistemska napaka in šum.

V kontekstu meritev v računalniških omrežjih sistemska napaka mnogokrat izvira iz invazivnosti meritev, šumnost meritev pa iz *neponovljivosti bremena*, ki ga merimo.

7.3.2 Naključnostno pogojene karakteristike meritev

Pri meritvah realnega prometa v omrežjih se moramo zavedati, da je to dogajanje do neke mere naključno pogojeno, kar pomeni, da pri ponavljanju meritev običajno ne pridemo do istih rezultatov. Slednje velja predvsem za meritve količine prometa v omrežjih. Glede na povedano moramo za doseganje objektivnejše slike o prometu izvesti večje število ponovitev meritev in nato rezultate ustrezno ovrednotiti (izračunati njihovo povprečje, upoštevati odklone, analizirati največje in najmanjše obremenitve v omrežju itd.).

7.3.3 Časovne karakteristike meritev

Pri meritvah v omrežjih se soočamo s spremenljivostjo količine prometa v omrežjih. Slednja je lahko stohastično ali naključno pogojena, deloma pa je tudi odraz nekih *časovnih trendov*, pogojenih s tem, *kdaj* meritve izvedemo. Časovne trende lahko razvrstimo na naslednje skupine:

- *dnevni trendi*: po izkušnjah ponudnikov internetnih storitev dnevna krivulja prometa niha od nekih maksimalnih do nekih minimalnih vrednosti in je skozi čas tipična;
- *tedenski trendi*: po izkušnjah ponudnikov tedenska krivulja skozi delovne dni niha približno enakomerno, njene amplitude pa se tokom vikenda zmanjšajo;
- *sezonski trendi*: po izkušnjah ponudnikov je količina prometa poleti manjša, kot v ostalih mesecih;

Poleg predhodno naštetih časovno pogojenih trendov ne smemo pozabiti na *vremensko pogojene trende* (v primeru slabega vremena se običajno promet po omrežju poveča, propustnost omrežja pa se zaradi vremenskih vplivov predvsem v bakrenih ali brezžičnih optičnih omrežjih (angl. *free space optics* - FSO) zmanjša) in na *dogodkovno pogojene trende*. Slednje lahko pričakujemo ob porajanju izrednih dogodkov (globalne nesreče, prenosi svetovnih športnih dogodkov itd.).

7.4 Primer orodja za izvajanje in analizo monitoringa podatkov iz omrežij

Primer brezplačno dostopnega orodja, ki ga za monitoring in analizo uporablja kar nekaj slovenskih ponudnikov internetnih storitev in upraviteljov omrežij, je razvojno okolje Cacti [17]. Slednje je temeljiteje opisano v delu [18]. Izdajanje zahtev po vzorcih in njihova dostava temelji na SNMP protokolu (angl. *simple network management protocol*), ki je opisan v dokumentu RFC 1157 in se nahaja z vidika TCP/IP protokola na aplikacijskem sloju. SNMP temelji na UDP protokolu, ki mora biti nameščen na vzorčeni aktivni napravi.

7.4.1 Koncept uporabe razvojnega okolja Cacti

Razvojno okolje Cacti deluje po konceptu konfiguriranja, pri čemer v fazi kofiguracije v sistem vnesemo mejne vrednosti, ki jih ne bi radi presegli (npr. velikost pomnilniške razpoke, zasedenost prometa, količino prometa na časovno enoto), sistem pa nam potem izrisuje v grafični obliki tok vzorčenih podatkov, v primeru preseženih vrednosti ali neželjenih dogodkov pa sistem uporabnika monitoringa alarmira vizuelno in v drugih oblikah (s pošiljanjem e-pošte, SMS sporočil itd.). Primere grafičnih predstavitev monitoriranja si bralec lahko ogleda v delu [18].

7.4.2 Strategija hrambe vzorčenih podatkov

Z rednim vzorčenjem v namene monitoriranja se soočamo s kopičenjem podatkov, ki jih za potrebe analiz (npr. napovedovanje trendov, iskanje vzročnih povezav med dogodki) običajno ponudniki in vzdževalci hranijo v podatkovnih bazah (PB). Osnovna značilnost tovrstnih podatkovnih baz je v tem, da so organizirane po konceptu *Round Robin Database*, kar pomeni, da je PB ciklična in ko se zapolni (je predhodno torej *omejena*), se novi vzorčeni podatki začnejo zapisovati čez najstarejše predhodno shranjene podatke. *Omejenost* običajno ne glasi na količino podatkov, temveč na njihovo starost. Tako večina omrežnih monitoring sistemov hrani vzorčene podatke stare do enega leta, starejše podatke pa nadomešča z novo pridobljenimi. S tem razrešimo problem kopičenja podatkov, delno pa smo na ta način soočeni z izgubljanjem podatkov za morebitne analize dogajanj na daljša časovna obdobja. Na tem mestu bi bilo smiselno vsaj del starejših vzorčenih podatkov prenašati v *arhivske podatkovne baze*.

7.5 Določanje kvantitativnih vrednosti opazovanih spremenljivk brez možnosti meritev

- določanje tipov in števila uporabnikov (posl., privatni) - določanje vrste storitev

Odkod dobiti te podatke - operaterji skrivajo!!! Profil uporabnika!

Poglavje 8

Modeliranje računalniških omrežij z orodjem OMNeT++

8.1 Uvod

Orodje OMNeT++ predstavlja odprtokodno simulacijsko ogrodje, ki temelji na diskretnemu proženju simulacijskih dogodkov [19, 20, 21]. Orodje je sestavljeno iz knjižnic napisanih v programskem jeziku C++, razvojnega okolja (angl. *Integrated Development Environment*) in okolja za poganjanje simulacij. OMNeT++ nudi osnovna orodja za postavljanje simulacijskih modelov, njihovo simuliranje in zbiranje ter analizo simulacijskih rezultatov. OMNeT++ torej ni klasični simulator temveč ponuja infrastrukturo za pisanje in poganjanje simulacij. Poleg osnovnega ogrodja je bilo v okviru drugih projektov kot je npr. INET [22] razvitih že veliko število paketov z že izdelanimi simulacijskimi moduli. Ker je orodje OMNeT++ zasnovano modularno, lahko že razvite simulacijske module ponovno uporabimo v svojih simulacijskih modelih. Obstoječe module lahko po eni strani uporabimo nespremenjene, po drugi strani pa je možno njihovo delovanje prilagoditi uporabnikovim potrebam. Sledeče poglavje predstavlja osnove orodja OMNet++ in delo z njim demonstrira na preprostih zgledih s področja modeliranja računalniških omrežij.

8.2 OMNeT++ osnovni gradniki in njihovo delovanje

Najvišja komponenta simulacijskega modela je t.i. *omrežje* (angl. *network*), ki je sestavljeno iz *modulov* in povezav med njimi. Izgradnja simulacijskega modela poteka na medsebojnemu povezovanju modulov, pri čemer le-ti komunicirajo

s pošiljanjem *sporočil* (angl. *messages*). Uporabnik lahko pri izgradnji svojega modela uporablja že obstoječe module ali pa ustvari svoje. Osnovni simulacijski moduli so na voljo že znotraj orodja OMNeT++, veliko število modulov pa je dostopnih na spletu. Primer zbirke takih modulov, ki implementirajo funkcionalnosti različnih omrežnih protokolov kot so na primer UDP, TCP, IP in IPv6, je ogrodje INET (angl. *INET Framework*) [22]. V orodju OMNeT++ obstajata dve skupini modulov, in sicer *enostavni* (angl. *simple modules*) in *sestavljeni moduli* (angl. *compound modules*). Sestavljeni moduli vsebujejo enega ali več enostavnih ali drugih sestavljenih modulov, pri čemer število gnezdenj ni omejeno. Za enostavnimi moduli stoji izvedljiva C++ programska koda, ki določa njihovo funkcionalnost. Enostavni moduli izhajajo iz razreda `cSimpleModule`.

Moduli med seboj komunicirajo s pošiljanjem *sporočil* (angl. *messages*) posredno preko *vrat* (angl. *gates*) in *povezav* (angl. *connections*) ali celo neposredno med moduli(redko). Obstajajo trije tipi vrat, in sicer *vhodna* (angl. *in*), *izhodna* (angl. *out*) in *dvosmerna* (angl. *inout*). Vrata so med seboj povezana preko povezav, ki jim lahko določamo posamezne lastnosti kot so *zakasnitev* (angl. *delay*), *propustnost* (angl. *data rate*) in *delež napak* (angl. *bit error rate*). Sporočila lahko poleg *časovne oznake* (angl. *timestamp*) vsebujejo poljubne podatkovne strukture.

Modulom lahko pripadajo *parametri* (angl. *parameters*), ki se uporabljajo za določanje delovanja preprostih modulov in nastavljanje topologije modelov. Osnovni gradniki in njihovo delovanje so razloženi v nadaljevanju poglavja.

Opisovanje simulacijskega modela v orodju OMNeT++ poteka na več nivojih z različnimi jeziki. V ta namen se uporabljajo različni datotečni tipi, in sicer:

- NED datoteke: z njimi opisujemo topologijo in komponente modela.
- C++ datoteke: vsebujejo programsko kodo, ki opisuje delovanje preprostih modulov in kanalov. Ločimo dva tipa C++ datotek, in sicer t.i. *zglavne datoteke* (angl. *source files*) s končnico `h`, ki vsebujejo deklaracije lastnosti in metod in *izvorne datoteke* (angl. *source files*) s končnico `cc`, ki vsebujejo implementacijo metod in s tem določajo delovanje komponent.
- INI datoteke: predstavljajo konfiguracijske datoteke, ki določajo nastavitve za izvajanje simulacij. Ponavadi je to zgolj ena datoteka, in sicer `omnetpp.ini`.

8.3 Opisovanje v jeziku NED

Jezik NED (*Network DEscription*) služi opisovanju topologije in komponent modela. Podajamo ga v datotekah s končnico `ned`. Z njim ne določamo samega delovanja komponent, ampak zgolj njihovo strukturo in lastnosti. Jezik NED se najpogosteje uporablja za definicije omrežij, definicije sestavljenih modulov in deklaracije enostavnih modulov.

Na začetku vsake NED datoteke določimo enolično ime gradnika, ki ga datoteka predstavlja na sledeč način:

- `network ime_omrezja`: za omrežje,
- `module ime_modula`: za sestavljen modul,
- `simple ime_modula`: za enostaven modul.

Samo strukturo in lastnosti gradnika določamo znotraj različnih sekcij NED opisa. Pomembnejše sekcije, ki jih lahko uporabimo v NED datotekah so razložene v nadaljevanju.

8.3.1 Parameteri

Sekcija `parameters` služi določanju parametrov komponentam. Vsakemu parametru dodelimo enolično ime, podatkovni tip in opcijsko njegovo privzeto vrednost ali porazdelitev. Parametru lahko določimo `volatile` način delovanja, kar pomeni, da se vrednost parametra evaluira vsakič, ko simulacijsko okolje parameter zahteva. Funkcionalnost `volatile` je pomembna takrat, ko parameter nima fiksne vrednosti, ampak zanj podamo določeno porazdelitev. Primer sekcije `parameters`:

```
parameters:
 // parameter a tipa integer s privzeto vrednostjo 1
 int a = default(1);

 // volatile parameter b tipa double
 // porazdeljen uniformno na intervalu [0,3]
 volatile double b = default(uniform(0,3);

 // parameter c brez privzete vrednosti
 int c;
```

8.3.2 Podmoduli

Sekcija `submodules` služi določanju vgnezdenih modulov. Podmodule lahko določamo le v NED datotekah omrežij in sestavljenih modulov (t.j. modulov, ki niso enostavni). Pri vsakem podmodulu določimo njegovo ime (ime modula, ki že obstaja v našem projektu) in enolično labelo. Ker ima vsak podmodul svojo labelo, lahko uporabimo več instanc istega modula. Primer sekcije `submodules`:

```
submodules:
 computer1: computer {
  @display("p=50,70");
 }
 computer2: computer {
  @display("p=202,70");
 }
 switch: switch {
  @display("p=386,70");
 }
```

Pri tem lastnost `@display` določa izgled gradnika v grafično uporabniškem vmesniku (GUI) (glej poglavje 8.3.7).

8.3.3 Vrata

Sekcija `gates` služi določanju vrat enostavnih in sestavljenih modulov (NED datoteke omrežij vrat nimajo). Pri vsakih vratih podamo njihov tip (`input`, `output` ali `inout`) in enolično ime. Definiramo lahko tudi vektorski tip vrat. Primer sekcije `gates`:

```
gates:
 // izhodna vrata o1
 output o1;

 // vektorska izhodna vrata o2 velikosti 3
 output o2[3];

 // vhodna vrata i1
 input i1;

 // vektorska vhodna vrata i2
 // velikost se bo dolocila kasneje
 input i2[];
```

8.3.4 Tipi

Sekcija `types` služi deklaraciji tipov, ki jih lahko uporabljamo znotraj NED datoteke. Primer sekcije `types` znotraj katere deklariramo kanal (glej razdelek 8.3.6):

```
types:
 channel C extends ned.DatarateChannel {
  datarate = 100Mbps;
 }
```

8.3.5 Povezave

Sekcija `connections` služi določanju povezav med vgnezdenimi moduli. Povezave lahko določamo le v NED datotekah omrežij in sestavljenih modulov (t.j. modulov, ki niso enostavni). Pri vsaki povezavi podamo

- vrata, ki so preko povezave med seboj povezana: `ime_modula.ime_vrat`,
- usmerjenost povezave: `-->`, `<--` ali `<-->`,
- če želimo, da povezava vsebuje kanal (glej razdelek 8.3.6), le-tega podamo znotraj povezave: `--> ime_kanala -->`.

Primer sekcije `connections`:

```
connections:
 // lokalna vrata nimajo predpone
 // vektorska vrata referenciramo z oglatimi oklepaji
 i1 --> computer1.o2[0];

 computer1.o2[1] <-- i2[0];

 // vektorska vrata lahko avtomatsko povecamo z ++
 i2++ --> computer1.o2[2];

 // povezava dveh vrat, ki niso lokalna
 computer2.i1 <-- computer1.o1;

 // povezava dveh vrat preko kanala C
 switch.o1 --> C --> computer2.i2++;
```

8.3.6 Kanali

Kanali niso sekcije same po sebi, ampak v sekcijah zgolj nastopajo. S pomočjo kanalov lahko določamo lastnosti povezav med moduli. V ta namen lahko uporabljamo tri tipe kanalov, ki so razloženi v nadaljevanju. Za primer uporabe kanalov znotraj povezave glej razdelek 8.3.5, za primer deklaracije kanala kot tip pa razdelek 8.3.4.

Kanali tipa `IdealChannel`

Kanali tipa `IdealChannel` se uporabljajo brez parametrov in služijo simuliranju idealnega prenosa paketov, t.j. brez napak, brez zakasnitev in z neskončno propustnostjo. Če v povezavi ne podamo kanala, se privzeto uporabi idealni kanal.

Kanali tipa `DelayChannel`

Kanali tipa `DelayChannel` se uporabljajo za simuliranje zakasnitev povezave in predstavljajo razširitev tipa `IdealChannel`. Modelirana zakasnitev se pri prenašanju običajnih sporočil ne upošteva. Upošteva se le pri prenašanju paketov (glej razdelek 8.4.1). Tem kanalom lahko določamo dve lastnosti, in sicer:

- `delay`: določa zakasnitev v enotah `s`, `ms`, `us`,...
- `disabled`: ali je kanal omogočen - če postavimo to lastnost na `True`, bo kanal zavrgel vsa sporočila.

Kanali tipa `DatarateChannel`

Kanali tipa `DatarateChannel` predstavljajo razširitev tipa `DelayChannel` in dodatno omogočajo simuliranje pasovne širine povezave in verjetnost napake

paketov pri prenosu. Modelirana pasovna širina in verjetnost napake se pri prenašanju običajnih sporočil ne upošteva. Upošteva se le pri prenašanju paketov (glej razdelek 8.4.1). Kanalom tipa `DelayChannel` lahko dodatno določamo tri lastnosti, in sicer:

- `datarate`: določa pasovno širino kanala v enotah `Kbps`, `Mbps`,... Pasovna širina se upošteva pri računanju časa prenosa paketa in se prišteje zakasnitvi pri prenosu (angl. *delay*). Vrednost 0 pomeni neomejeno pasovno širino.
- `ber` (*Bit Error Rate*): določa verjetnost napake na nivoju enega bita. Lahko zavzame vrednosti znotraj intervala $[0, 1]$. Če simulacijsko okolje ugotovi, da je prišlo do napake, postavi zastavico `Error flag` na nivoju paketa.
- `per` (*Packet Error Rate*): določa verjetnost napake na nivoju enega paketa. Lahko zavzame vrednosti znotraj intervala $[0, 1]$. Če simulacijsko okolje ugotovi, da je prišlo do napake, paketu postavi zastavico `Error flag`.

8.3.7 Lastnosti

Kot kanali, tudi lastnosti niso sekcije same po sebi, ampak v sekcijah zgolj nastopajo. Z njimi lahko v NED datotekah določimo vrednosti lastnostim (angl. *properties*) posameznim objektom (modulom, parametrom, vratom, povezavam itd.). Lastnosti referenciramo z oznako `@`, za katero podamo ime lastnosti, ki ji sledi njena vrednost znotraj oklepaja. Primeri pogostejše uporabljenih lastnosti so:

- `@display`: določa izgled gradnika v GUI. Primer uporabe: `@display ("i=block/browser");`
- `@unit`: določa enote parametra. Primer uporabe: `double startTime @unit(s) = default(0 s);`
- `@statistic`: za beleženje statistike preko signalov (glej poglavje 8.6).

8.4 Programiranje v jeziku C++

Delovanje preprostih modulov in kanalov določimo v jeziku C++. V tem delu bo opisano zgolj programiranje preprostih modulov, saj podrobnosti programiranja kanalov presegajo obseg tega dela. Programiranje kanalov je med drugim opisano v [21].

Programiranje v orodju OMNeT++ temelji na dogodkovnem proženju metod - t.i. dogodkovno vodeno programiranje (angl. *event-driven programming*), pri čemer so glavni dogodki vezani na sporočila, ki jih modul dobi bodisi od sebe bodisi od drugih modulov. Delovanje posameznega modula opisujemo z dvema datotekama, in sicer

- datoteka `ime_modula.h`: vsebujejo deklaracije metod in lastnosti ter reference na knjižnice. Vedno je potrebno vključiti knjižnico `<omnetpp.h>`.

- datoteka `ime_modula.cc`: vsebuje implementacijo metod. Datoteka mora vključevati datoteko `ime_modula.h`.

V nadaljevanju sledi opis osnovnih razredov v okolju OMNeT++, osnovnih metod, s katerimi lahko programiramo obnašanje enostavnih modulov in posebnosti C++ programiranja v okolju OMNeT++.

8.4.1 Osnovni razredi v okolju OMNeT++

Sledeči razdelek opisuje nekatere osnovne razrede v okolju OMNeT++, katerih deklaracije so podane v zglavni datoteki `<omnetpp.h>`.

Razred `cSimpleModule`

Enostavni moduli se vedno dedujejo iz razreda `cSimpleModule`. Modul ima poleg konstruktorja in destruktorja že deklarirane virtualne metode, ki jih lahko uporabnik implementira in se kličejo ob določenih simulacijskih dogodkih. To so `initialize()`, `handleMessage(cMessage *msg)` in `finish()` (glej razdelek 8.4.2).

Razred `cMessage`

Razred `cMessage` služi implementaciji sporočil, ki si jih moduli med seboj izmenjujejo. Z njim so povezane sledeče metode, ki jih lahko uporabljamo znotraj programske kode - predvsem v metodi `handleMessage()` (glej razdelek 8.4.2):

- `send()`: za pošiljanje sporočil drugim modulom preko specificiranih vrat; primer: `send(msg,"out")`.

- `sendDelayed()`: za zakasnjeno pošiljanje sporočil drugim modulom preko specificiranih vrat; primer `sendDelayed(msg,0.005,“out”)`.

- `scheduleAt()`: za pošiljanje sporočil samemu sebi. Primer uporabe: generator prometa pošilja sporočila samemu sebi ob intervalih, ki so definirani z medprihodnimi časi. Sprejem tega sporočila povzroči generiranje prometa; primer: `scheduleAt(simTime()+par("interArrivalTime"). doubleValue(), msg)` (glej razdelek 8.4.3);

- `cancelEvent()`: za preklic dogodka, ki je bil prožen z metodo `scheduleAt()`; primer: `cancelEvent(msg)`.

- `cancelAndDelete()`: za preklic in brisanje dogodka.

Razred `cGate`

Razred `cGate` omogoča delo z vrati modula. Instance tega razreda vedno pripadajo določenemu modulu, ki z njimi tudi upravlja (klic konstruktorja in destruktorja). Glavne metode za delo z vrati so

- `gate()`: vrne referenco na vrata z imenom, ki je podan kot argument metode; primer: `cGate *outGate = gate("g")`. Pri `inout` vratih imamo za vsako smer svoj objekt: `cGate *inGate = gate("g$i"); cGate outGate = gate("g$o")`.
- `hasGate()`: vrne `True`, če modul ima vrata s podanim imenom; primer `if (hasGate("gOut")) ...`
- `gateSize()`: vrne velikost vektorskih vrat, ki so podana kot argument; primer: `gateSize("g")`.

Razred `cChannel`

Podobno kot pri programiranju enostavnih modulov tudi pri programiranju kanalov izhajamo iz obstoječega OMNeT++ razreda, in sicer iz razreda `cChannel`. Ponavadi potrebe po pisanju lastnih kanalov ni, saj lahko izhajamo iz predefiniranih kanalov (`IdealChannel`, `DelayChannel` in `DatarateChannel`), zato se v podrobnosti programiranja kanalov ne bomo spuščali, vseeno pa je v nadaljevanju razložena osnovna uporaba kanalov v jeziku C++.

Kanali so vedno vezani na vrata. Do njih torej dostopamo zgolj preko vrat z uporabo metode `getChannel()`. Primer uporabe: `cChannel *channel = gate->getChannel()`. Lahko pristopamo tudi v obratni smeri, tako da preko kanala pridemo do reference vrat: `cGate *gate = channel->getSourceGate()`.

Kanali, ki modelirajo zakasnitev paketov pri prenosu pravimo *prenosni kanali* (angl. *Transmission Channels*). Preko podanih vrat lahko pridemo do njihovega prenosnega kanala (ni nujno, da je to ravno kanal, ki je neposredno vezan na ta vrata) na sledeč način: `cChannel channel = gate->getTransmissionChannel()`.

Pakete lahko preko prenosnega kanala prenašamo, šele ko je le-ta prost. Zasedenost prenosnega kanala lahko preverimo z metodo `isBusy()`, do časa sprostitve prenosnega kanala pa pridemo z metodo `getTransmissionFinishTime()`.

Razred `cPacket`

Objekti, ki pripadajo razredu `cPacket`, t.i. *paketi*, predstavljajo razširitev objektov razreda `cMesssage` in omogočajo simuliranje prenosa komunikacijskih paketov. Paketi omogočajo modeliranje zakasnitve prenosa, omejene pasovne širine prenosnih kanalov in napak pri prenosu preko kanalov. V ta namen se uporabljajo skupaj z objekti tipa `cChannel`, ki jih lahko vežemo na povezave med moduli (glej razdelek 8.3.6). Pogosto uporabljene metode razreda `cPacket` so

- `getBitLength(), getByteLength()`: vrne dolžino paketa v bitih oziroma bajtih, ki se uporabi za izračun trajanja prenosa paketa preko kanala,
- `setBitLength(), setByteLength()`: metodi za nastavljanje dolžine paketa, pri čemer argument podamo v bitih oziroma v bajtih,
- `setBitError(), hasBitError`: vračanje oziroma nastavljanje zastavice, ki pove, da je paket pokvarjen,
- `getDuration()`: vračanje časa prenosa paketa - vrednost je pravilna šele po opravljenem prenosu paketa.

Razred `cQueue`

Razred `cQueue` predstavlja implementacijo čakalne vrste. Privzeto delovanje čakalne vrste je *First In First Out* (FIFO), ki pa ga lahko spremenimo. Spremenimo jo lahko tudi v prioritetno čakalno vrsto. Pogosto uporabljene metode razreda `cQueue` so

- `insert()`: metoda za dodajanje elementov v čakalno vrsto,
- `pop()`: metoda za odstranjevanje elementov iz čakalne vrste (vsakič se vzame tisti element, ki je naslednji na vrsti),
- `clear()`: metoda, ki izprazni čakalno vrsto,
- `isEmpty()`: metoda, ki vrne odgovor na vprašanje - ali je čakalna vrsta prazna,
- `length()`: metoda, ki vrne dolžino čakalne vrste.

8.4.2 Osnovne metode enostavnih modulov v OMNeT++

Sledeči razdelek opisuje osnovne metode, ki jih programiramo kot odzive na določene simulacijske dogodke.

Metoda `ime_modula()`

Metoda `ime_modula()` predstavlja konstruktor, ki se kliče ob kreaciji objekta, ki modul predstavlja. Metodo kliče simulator.

Metoda `initialize()`

Metoda `initialize()` je inicializacijska metoda, ki jo simulator kliče še pred prvim simulacijskim dogodkom. Če je to potrebno, iz te metode sprožimo prvi simulacijski dogodek.

Metoda `handleMessage(cMessage *msg)`

Metoda `handleMessage` predstavlja jedro vsakega modula in s tem tudi simulacije, saj izmenjava sporočil predstavlja glavne simulacijske dogodke v okolju OMNeT++. Metoda se sproži vsakič, ko modul dobi sporočilo - bodisi od sebe bodisi od drugih modulov. Ob sprejemu sporočila se mora metoda ustrezno odzvati glede na njegovo vsebino. Odziv na sprejem sporočila ponavadi vsebuje vsaj pošiljanje novega sporočila bodisi sebi bodisi nekemu drugemu modulu.

Metoda `finish()`

Metoda se kliče, ko v čakanju ni več nobenih dogodkov - ob koncu simulacije. Ponavadi se simulacija zaključi, ko na čakanju ni nobenega sporočila več v nobenem modulu. Kliče se samo v primeru, ko se je simulacija izvedla brez napak. Iz tega razloga ta metoda ni primerna za čiščenje pomnilnika - to je potrebno narediti v destruktorju. Metoda `finish` je primerna predvsem za obdelavo pridobljenih podatkov po izvedbi simulacije.

Metoda `~ime_modula()`

Metoda `~ime_modula()` predstavlja desktruktor, ki se kliče ob uničenju objekta. V metodi je potrebno izbrisati vse objekte, ki smo jih ustvarili z `new`. Za sporočila, ki so trenutno na čakanju (angl. *pending*) uporabimo metodo `cancelAndDelete()`, za ostale objekte pa metodo `delete()`. Simulacijsko okolje nas bo ob koncu simulacije opozorilo, če za seboj nismo počistili.

8.4.3 Ostale posebnosti pri programiranju

Sledeči razdelek izpostavi nekaj posebnosti pri C++ programiranju v okolju OMNeT++.

Registracija modula

Na začetek datoteke `ime_modula.cc` moramo vedno vključiti makro `Define_Module(ime_modula)`, ki modul registrira v OMNeT++ okolju. V nasprotnem primeru nam simulacijsko okolje javi napako, da modul ne obstaja. V primeru programiranja lastnih kanalov le-te registriramo na podoben način z makrojem `Define_Channel(ime_kanala)`.

Dostopanje do parametrov modula

Znotraj NED datotek lahko definiramo parametre modula, preko katerih se lahko posredujejo določene vrednosti do najnižjega nivoja, t.j. do enostavnih modulov in kanalov. Do vrednosti parametrov lahko pridemo z uporabo metode `par()`; primer uporabe: `double delay = par("delay")`. Parametri tipa `volatile` se bodo ovrednotili vsakič, ko bomo zahtevali njihovo vrednost preko te metode, ostali parametri pa bodo cel čas simuliranja imeli enako vrednost.

Klic metod drugih modulov

Metode, ki pripadajo različnim modulom, se privzeto med seboj vidijo. V ta namen je v vsako metodo, ki jo želimo klicati od zunaj, t.j. iz nekega drugega modula, na začetek potrebno dodati bodi makro `Enter_Method()`, pri katerem se bo ob klicu metode v GUI vmesniku pojavil podani izpis, bodisi makro `Enter_Method_Silent()`. Primer uporabe makroja: `Enter_Method ("release(%ld)", amountToRelease)`.

Opazovanje vrednosti spremenljivk med simulacijo

Če želimo opazovati vrednosti posamezne spremenljivke med izvajanjem simulacije, to povemo z makrojem `WATCH()` v metodi `initialize()`. Primer uporave: `WATCH(ime_spremenljivke)`.

Izpisovanje v konzolo

Med izvajanjem simulacije lahko v konzolo izpisujemo nize na sledeč način
`EV « niz1 « niz2 « spremenljivka1 « ...;`

Izpisovanje v GUI simulatorja

Izpisovanje v GUI prožimo z metodo `getDisplayString().setTagArg()`.
Primer uporabe:

```
char buf[80];
sprintf(buf, "oznaka: %d\n",spremenljivka);
getDisplayString().setTagArg("t",0, buf);
```

8.5 Konfiguriranje eksperimentov z INI datotekami

Medtem ko na NED datoteke gledamo kot na del modelov, INI datoteke (ponavadi je na projekt vezana samo ena datoteka, in sicer `omnetpp.ini`) predstavljajo del eksperimentov. Tak način dvonivojskega opisovanja nam omogoča ločitev postavljanja modelov od eksperimentov. INI datoteke torej opisujejo eksperimente, med katerimi lahko izbiramo po zagonu simulacije. Posamezna INI datoteka lahko vsebuje več t.i. konfiguracij (angl. *configurations*) parametrov nad istim ali različnimi modeli (t.j. omrežji). Ob zagonu simulacije izberemo konfiguracijo, ki jo želimo simulirati. V nadaljevanju so navedene nastavitve, ki jih najpogosteje določamo v INI datoteki.

- Model oziroma omrežje nad katerim želimo izvajati simulacije določamo z lastnostjo `network`, ki mu dodelimo ime NED datoteke izbranega modela; primer: `network=FifoNet`.

- Vrednosti parametrov lahko v INI datotekah določamo samo tistim parametrom, katerih vrednost še ni bila določena v NED datotekah. Določamo jih tako, da na levi strani podamo ime parametra v obliki `ime_omrezja.ime_modula.ime_parametra`, na desni strani pa njegovo vrednost (če je potrebno vključno z enotami). Vrednost je lahko tako kot v NED datotekah podana kot porazdelitev. Če je modul vgnezden, je potrebno podati celo pot do njega. Pri podajanju poti lahko uporabljamo tudi t.i. *nadomestne znake* (angl. *wildcards*), npr. `*`, `**` in `?`. Primer: `**.fifo.bitsPerSec = 1000bps`. Orodje nam omogoča konfiguracijo več simulacijskih ponovitev z različnimi vrednostmi parametrov ali različnimi omejitvami.

  Primer 1:

  ```
  *.ime_parametra = ${1, 2, 5, 10..50 step 10}
  ```

  Primer 2:

  ```
  *.ime_parametra = ${N=1, 2, 5, 10..50 step 10}
  ```

  Primer 3:

  ```
  *.param1 = ${i=1..10 step 2}
  *.param2 = ${j=1..20 step 3}
  constraint = $j <= sqrt($i)
  ```

- Običajno je potrebno določiti časovne parametre simulacij, predvsem čas njihovega trajanja. Primer: `sim-time-limit = 100h`.

- Nastavljanje načinov zbiranja simulacijskih podatkov je razloženo v poglavju 8.6.

OMNeT++ INI datoteke so v grobem sestavljene iz dveh tipov sekcij, in sicer iz sekcije *General* in iz *Config* sekcij.

V sekciji *General* konfiguriramo splošne nastavitve, ki bodo pri vseh eksperimentih enake. S tem se izognemo ponovitvam istih nastavitev v vseh sekcijah. Primer sekcije *General*:

```
[General]
network = FifoNet
sim-time-limit = 100h
```

Sekcij *Config* je lahko več. V primeru, da želimo definirati le en eksperiment, sekcij *Config* ne potrebujemo, saj lahko eksperiment opišemo zgolj s sekcijo *General*. Primer dveh sekcij *Config*, ki sledita gornji sekciji *General*:

```
[Config Fifo1]
description = "low job arrival rate"
**.gen.sendIaTime = exponential(0.2s)
**.gen.msgLength = 100b
**.fifo.bitsPerSec = 1000bps
```

```
[Config Fifo2]
description = "high job arrival rate"
**.gen.sendIaTime = exponential(0.01s)
**.gen.msgLength = 10b
**.fifo.bitsPerSec = 1000bps
```

8.6 Zbiranje simulacijskih rezultatov

Simulacijske rezultate lahko po eni strani pridobivamo že med tekom simulacije, t.i. *runntime analiza* (glej razdelek 8.4.3). Tak način zbiranja podatkov pa ni primeren za obsežnejše simulacije. V ta namen moduli v okolju OMNeT++ uporabljajo t.i. *signale*, ki jih lahko uporabljamo bodisi nad enostavnimi moduli bodisi na kanali. Moduli in kanali preko signalov oddajajo določene vrednosti (npr. čas nahajanja zahteve v čakalni vrsti, dolžina čakalne vrste itd.), ki jih simulacijsko okolje zbira na podlagi podanih konfiguracij. Odvisno od konfiguracije se te vrednosti po potrebi še dodatno obdelajo v metodi `finish()` - *skalarni podatki* ali pa shranijo v vektor - *vektorski podatki*. Rezultati simuliranja se shranijo v izhodne datoteke skalarjev (angl. *output scalar files*) in izhodne datoteke vektorjev (angl. *output vector files*). Vsak tek simulacije (angl. *simulation run*) rezultate shrani v svojo datoteko.

Uporabo signalov mora uporabnik konfigurirati na treh nivojih, in sicer:

- v izvorni in zglavni datoteki enostavnega modula (C++),
- v NED datoteki in
- v INI datoteki.

V nadaljevanju je podrobneje opisana uporaba signalov.

8.6.1 Programiranje signalov (C++)

Do signalov v okolju OMNeT++ dostopamo preko njihovih ID-jev (*signalID*), ki jih shranjujejmo v spremenljivke tipa `simsignal_t`. Signale, ki jih nameravamo uporabiti je potrebno deklarirati v zglavni datoteki modula oziroma kanala

```
simsignal_t imeSignala;
```

V `initialize()` metodi modula, ki mu signal pripada, posamezen signal registriramo z metodo `registerSignal()`, ki ji kot argument podamo ime signala, preko katerega bo le-ta viden simulacijskemu okolju

```
imeSignala = registerSignal("imeSimSignala");
```

Signali objekt je tako pripravljen za oddajanje vrednosti simulacijskemu okolju, ki jih prožimo z metodo `emit()`. Oddajanje signala praviloma prožimo vsakič, ko se opazovana vrednost spremeni. Metodi `emit()` kot argument podamo ime signalnega objekta (ime spremenljivke) in vrednost, ki jo oddajamo preko signala

```
emit(imeSignala, vrednost);
```

Čeprav se ponavadi oddajajo številske vrednosti, je lahko oddana vrednost predstavljena tudi z drugimi podatkovnimi tipi.

V nadaljevanju sledi primer uporabe signala, ki oddaja dolžino čakalne vrste. Najprej deklariramo signal `queueLengthSignal`

```
simsignal_t queueLengthSignal;
```

V metodi `initialize()` signal registriramo

```
queueLengthSignal = registerSignal("queueLength");
```

Vsakič, ko se dolžina čakalne vrste spremeni (bodisi pride nova zahteva bodisi se v procesiranje vzame naslednja iz čakalne vrste), je potrebno javiti novo dolžino čakalne vrste (predpostavljamo, da za čakalno vrsto uporabljamo objekt `queue`, ki pripada razredu `cQueue`)

```
emit(queueLengthSignal,queue.length());
```

8.6.2 Konfiguracija beleženja signalov (NED)

Avtomatsko beleženje signala vklopimo z deklaracijo v NED datoteki enostavnega modula ali kanala, ki mu signal pripada. To naredimo v sekciji `parameters` z uporabo lastnosti `@statistic`, pri čemer ime signala podamo kot *indeks lastnosti* (angl. *property index*) v oglatih oklepajih, lastnosti beleženja signala pa kot *vrednosti lastnosti* (angl. *property values*) v navadnih oklepajih:

```
parameters:
   @statistic[imeSimSignala](lastnosti beleženja signala);
```

Lastnosti beleženja, ki jih ponavadi nastavljamo so:

- `title`: opis pomena signala, ki ga lahko uporabijo razna vizualizacijska orodja pri prikazovanju,
- `unit`: enote - za vizualizacijo,
- `interpolationmode`: uporabljena metoda interpolacije, če je le-ta potrebna,
- `enum`: simbolna imena za različne numerične vrednosti, npr. `"IDLE=1, BUSY=2, DOWN=3"`,
- `record`: način izračunavanja skalarnih vrednosti iz vseh pridobljenih vrednosti signala.

Pogosteje uporabljene vrednosti za lastnost `record` so:

- `vector`: vse vrednosti se shranijo v vektor,

- `count`: prešteje število oddanih signalov - vsebina signalov v tem primeru ni relevantna,
- `last`: vzame samo zadnjo oddano vrednost,
- `sum`: sešteje vse oddane vrednosti,
- `mean`: vzame srednjo oddano vrednost,
- `min`: vzame minimalno oddano vrednost,
- `max`: vzame maksimalno oddano vrednost,
- `timeavg`: s pomočjo časovnih oznak izračuna časovno povprečje oddanih vrednost,
- `stats`: vzame `count`, `sum`, `mean`, standardno deviacijo deviation, `min` in `max`,
- `histogram`: vzame `stats` in histogram vrednosti.

Primer:

```
PRIMER
```

8.6.3 Konfiguracija beleženja signalov (INI)

V INI datotekah lahko določamo način beleženja podatkov na nivoju posameznih simulacij (konfiguracij). Ponavadi nastavljamo sledeče parametre:

- `output-vector-file`: datoteke za shranjevanje vektorskih podatkov (privzeto: `$resultdir/$configname-$runnumber.vec`),
- `output-scalar-file`: datoteke za shranjevanje skalarnih podatkov (privzeto: `$resultdir/$configname-$runnumber.sca`),
- `vector-recording`: vklop/izklop beleženja vektorskih podatkov na nivoju signala, modula ali celotne simulacije; primer: `**.vector-recording = true/false`,
- `scalar-recording`: vklop/izklop beleženja skalarnih podatkov na nivoju signala, modula ali celotne simulacije; primer: `**.scalar-recording = true/false`,
- `result-recording-mode`: način beleženja podatkov na nivoju signala, modula ali celotne simulacije, pri čemer lahko izbiramo med vrednostmi parametra `record` lastnosti `@statistic` v NED datotekah (glej razdelek 8.6.2); pri tem se vrednost definirana v NED datoteki povozi; primer: `**.queueLength.result-recording-mode = timeavg,max`,

- `param-record-as-scalar`: vklop/izklop shranjevanja parametrov v datoteko skalarjev na nivoju modula ali celotne simulacije; primer: `**.networkLoad.param-record-as-scalar = true`,
- `sim-time-limit`: določitev časa simuliranja; primer: `sim-time-limit = 10000 s`,
- `warmup-period`: začetni čas simuliranja, ko naj se statistike ne beležijo; primer: `warmup-period = 100 s`,
- `record-eventlog`: vklop/izklop beleženja dogodkov (angl. *eventlog*); `record-eventlog = true/false`.

8.7 Simuliranje

Način poganjanja simulacij lahko nastavljamo preko okna *Run Configurations*, do katerega pridemo preko menija *Run*. V nadaljevanju so opisane možnosti za simuliranje in anlaizo simulacijskih rezultatov v orodju OMNeT++.

8.7.1 Okolje *Tcl/Tk*

Preden začnemo z beleženjem podatkov je priporočljivo preveriti pravilnost delovanja modela preko simulacijskega okolja *Tcl/Tk*. To naredimo tako, da v oknu *Run Configurations* kot *User Interface* izberemo *Default* oziroma *Tcl/Tk*. Pri tem je pomembno, da je polje *Run number* prazno (privzeto). Izberemo lahko tudi INI datoteko (privzeta je *omnetpp.ini* in konfiguracijo. Če konfiguracije nismo izbrali in jih imamo v INI datoteki več, nas orodje *Tcl/Tk* najprej vpraša, katero konfiguracijo želimo simulirati. Simulator zatem vzpostavi konfiguracijo in določi parametre na sledeč način: če ima parameter vrednost določeno v NED datoteki, se mu dodeli ta vrednost, sicer preveri, če je njegova vrednost določena v INI datoteki. Če je, se vzame vrednost iz te datoteke, sicer mora uporabnik vnesti vrednost preko *Tcl/Tk* okna. Orodje *Tcl/Tk* omogoča spremljanje poteka simulacije preko *Inspect Network* vmesnika, v katerega se izrisuje pretok sporočil preko modulov ali preko konzole, v katerega se izpisujejo definirani izpisi. Ob dvojnem kliku na posamezen modul v *Inspect Network* oknu, se odpre okno z lastnostmi modula, kjer lahko spremljamo vrednosti njegovih parametrov in vrednosti spremenljivk definiranih v makroju `WATCH()` (glej razdelek 8.4.3).

8.7.2 Simuliranje preko ukazne vrstice

Okolje *Tcl/Tk* je primerno le za preverjanje pravilnosti delovanja modela in njegovo razhroščevanje (angl. *debugging*). Za zbiranje simulacijskih rezultatov imamo v okolju OMNeT++ na voljo simuliranje preko ukazne vrstice (angl. *Command Line*). V oknu *Run Configurations* v ta namen kot *User Interface* izberemo *Command Line*. Nastaviti je potrebno konfiguracijo, ki jo bomo simulirali in simulacijske teke (angl. *Run number(s)*), ki jih želimo simulirati (privzeto =

0). Za izvedbo vseh simulacijskih tekov, definiranih v INI datoteki, v polje *Run number* vnesemo *.

8.7.3 Obdelava simulacijskih rezultatov

Po uspešni izvedbi simulacij se podatki, katerih beleženje smo nastavili v NED in INI datotekah shranijo v direktorij *results* v datoteke tipa *vec* za vektorske podatke, tipa *sca* za skalarne podatke in *elog* za beleženje dogodkov. Posamezna datoteka je vezana na simulacijski tek - vsak simulacijski tek ima svojo *vec*, *sca* in *elog* datoteko. Če imamo posamezno beleženje izklopljeno, se datoteka ne ustvari.

Ob dvojnem kliku na datoteke tipa *vec* ali *sca* se odpre okolje *Analysis Editor*, v katerem lahko vsebino datotek pregledujemo, filtriramo rezultate in izrisujemo različne grafe. Ob dvojnem kliku na datoteke tipa *elog* se odpro okno, v katerem lahko pregledujemo potek simulacijskih dogodkov.

Literatura

[1] M. Anu, "Introduction to modeling and simulation," in *Proceedings of 1997 Winter Simulation Conference*, 1997.

[2] L. Kleinrock and R. Gail, *Queuing systems, problems and solutions.* John Wiley & Sons, 1996.

[3] N. Zimic and M. Mraz, *Temelji zmogljivosti računalniških sistemov.* Založba FE in FRI, 2006.

[4] N. C. Hock, *Queuing Modelling Fundamentals.* Joh Wiley & Sons, 1996.

[5] J. L. Peterson, *Petri Net Theory and the Modeling of Systems.* Prentice Hall, 1981.

[6] T. Murata, "Petri nets: Properties, analysis and applications," 1989.

[7] W. G. Schneeweiss, *Petri Nets for Reliability Modeling.* LiLoLe Verlag, 1999.

[8] N. Jensen and L. Kristensen, *Coloured Petri Nets.* Springer, 1998.

[9] K. Jensen, "A brief introduction to coloured petri nets," in *Proceedings of the Third International Workshop on Tools and Algorithms for Construction and Analysis of Systems (TACAS '97)*, 1997.

[10] A. S. Tanenbaum and D. J. Wetherall, *Computer Networks.* Prentice Hall, 2011.

[11] D. Božić, *Analiza in zgled uporabe programskega orodja CPNTools za postavljanje modelov dinamičnih sistemov.* Diplomsko delo FRI-UL, 2012.

[12] "CPNTools." `http://cpntools.org/`, 2012.

[13] M. Dolenc, *Verifikacija komunikacijskih protokolov na osnovi barvnih Petrijevih mrež.* Diplomsko delo FRI-UL, 2015.

[14] "Matematična ocena latence proizvajalca Rugged." `https://w3.siemens.com/mcms/industrial-communication/en/rugged-communication/Documents/AN8.pdf/`, 2015.

[15] "Matematična ocena latence proizvajalca O3b Networks." http://www.o3bnetworks.com/media/40980/white%20paper_latency%20matters.pdf/, 2015.

[16] "Internet dveh hitrosti." http://webfoundation.org/2015/10/net-neutrality-fails-to-load-web-foundation-response-to-todays-eu-vote//, 2016.

[17] "About Cacti." http://www.cacti.net//, 2016.

[18] P. Antončič, *Monitoriranje računalniških omrežij.* Diplomsko delo FRI-UL, 2012.

[19] "OMNeT++." www.omnetpp.org/, 2012.

[20] A. Varga, *Modeling and Tools for Network Simulation*, ch. OMNeT++, pp. 35–59. Springer-Verlag, 2010.

[21] A. Varga, *OMNeT++ User Manual, Version 4.1.* OpenSim Ltd., 2010.

[22] "INET." http://inet.omnetpp.org/, 2012.