1. ПРОТОКОЛИ И АРХИТЕКТУРЕ

Концепти дистрибуираног процесирања и рачунарских мрежа условљавају да целине у различитим системима имају потребу да међусобно комуницирају. Користимо термин целина[1] и систем у општем смислу. Пример целине су: кориснички апликациони програм, пренос датотека, системи за управљање базама података, уређаји за електронску пошту и терминали. Пример система су: рачунар, терминал и удаљени сензор. Уочимо да су у неком случају целина и систем коегзистентни. У општем случају целина је било шта способно да шаље и прима информације, а систем је физички одвојен објекат који садржи једну или више целина.

Две целине које треба да комуницирају „морају да говоре истим језиком". Шта комуницира, како се комуницира и када се комуницира мора да се повинује неком опште прихватљивом скупу конвенција целина, које су укључене у комуникацију. Скуп конвенција назива се протокол, и он се може дефинисати као скуп правила која управљају разменом података између две целине. Кључни елементи протокола су:

- синтакса*:* укључује формат података, кодирање и нивое сигнала;
- семантика*:* укључује управљачке информације за координацију и вођење рачуна о грешкама;
- временска синхронизација[2]: укључује подешавање брзина и секвенционирање.

Један од примера је често коришћени протокол је HDLC[3]. Подаци које треба разменити морају се слати у рамовима специфичног формата (синтакса). Управљачко поље обезбеђује различите регулаторске функције као што су постављање врсте рада и успостављање везе (семантика).

Резултат стандарда су следећи:

- произвођачи се подстичу да примене стандарде због очекивања да ће се, због широке употребе стандарда, њихови производи више продавати,
- корисници захтевају да стандарде примењују сви произвођачи јер могу опрему набављати од било кога од њих.

Задатак да се обезбеди комуникација између апликација на различитим рачунарима превише је комплексан да би се посматрао као једна целина. Проблем се мора раздвојити у целине које се могу лакше реализовати. Тако, пре него што неко развије стандарде, мора да постоји структура или архитектура која дефинише комуникационе циљеве.

---

[1] *Entity* - ентитет
[2] *Timing*
[3] *High Level Data Link Control*

1. Протоколи и архитектура

Интернационална организација за стандардизацију (ISO[1]) је 1977. године формирала комисију са задатком да развије такву архитектуру. Резултат је отворени систем за међусобно повезивање (OSI[2]), референтни модел који је оквир за дефинисање стандарда за повезивање хетерогених рачунара. OSI модел обезбеђује основу за повезивање отворених система дистрибуираних апликативних процеса. Термин „отворен" одређује способност да било која два система која подржавају референтни модел и одговарајуће стандарде могу међусобно да се повежу.

Мрежна архитектура

Због комплексности пројектовању многе мреже су организоване као низ слојева или нивоа. Број слојева, као и њихова имена разликују се од мреже до мреже. У свим мрежама намена сваког слоја је да пружи одређену услугу вишим слојевима, штитећи те нивое од детаља (нпр. како се услуга реализује).

Слој N на једном рачунару води конверзацију са слојем N на другом рачунару. Правила и конвенције које се користе у овој конверзацији заједничке су и познате као протокол N-тог слоја.

Практично, подаци се не шаљу директно из слоја N једног рачунара ка слоју N другог рачунара (осим најнижи слој). Уместо тога, сваки слој шаље податке и управљачке информације ка слоју који је одмах испод, докле год не стигне до најнижег слоја. На најнижем слоју постоји физичка комуникација са другим рачунаром, супротно од виртуелне комуникације коју користе виши слојеви. Између сваког пара суседних слојева постоји интерфејс. Интерфејс[3] слојева дефинише које основне операције и услуге нижи слој нуди вишем слоју N. Када пројектант мреже одлучи колико слојева ће укључити у мрежу и шта ће који од њих да ради, најважније је јасно дефинисати интерфејсе између слојева. Јасно дефинисање слојева, заузврат, захтева да сваки слој извршава специфични скуп јасно дефинисаних функција. Последица тога је да се може један слој заменити потпуно другим слојем (нпр. телефонска линија се може заменити сателитском везом), пошто је све што се тражи од новог слоја да пружи исти скуп услуга за слој изнад себе као што је претходно био обезбеђен.

Скуп слојева и протокола назива се слојевита архитектура.

Спецификација архитектуре мора да садржи довољно информација да омогући ономе ко њу имплементира да изгради програм за сваки слој, тако да програм исправно реализује одговарајући протокол. Ни детаљи примене, ни спецификација интерфејса нису делови архитектуре. У ствари, чак ни сви интерфејси на свим рачунарима нису исти, а ипак је обезбеђено да сваки рачунар може правилно да користи протокол.

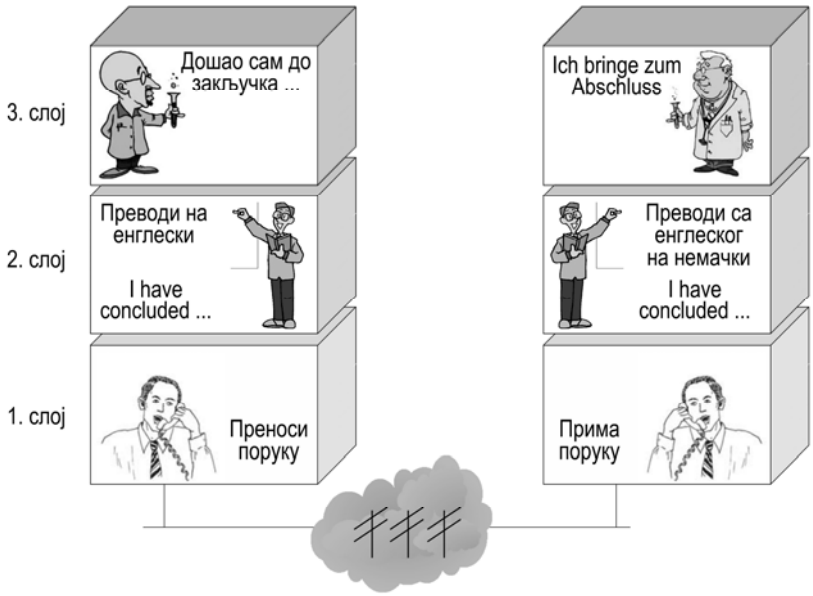
Пример вишеслојне архитектуре је следећи[4]: замислите да два истраживача (парњаци слоја 3) од којих је један у Београду а други у Минхену, желе да комуницирају (слика 1.1).

---

[1] *International Organization for Standardization*
[2] *Open System Interconnection*
[3] Користи се и термин сучеље.
[4] [1]

Пошто немају заједнички језик они укључују преводиоце (парњак - процес 2. слоја), а сваки од њих контактира инжењере (парњак- процес 1. слоја).

Истраживач 1 жели да пренесе своје закључке свом парњаку. Он шаље своје закључке на српском језику као поруке преко интерфејса 3. и 2. слоја, ка свом преводиоцу који реченицу „Дошао сам до закључка...” преводи у „*i have concluded....* ”, или на француски језик, зависно од протокола 2. слоја. Преводилац затим даје поруку свом техничару који поруку преноси телеграмом, телефоном, рачунарском мрежом или неким другим средством, зависно како су се унапред договорили (протокол 1. слоја). Када порука стигне она се преводи на немачки и преноси преко интерфејса 2. и 3. слоја истраживачу 2. Уочимо да је сваки протокол комплетно независан од других докле год се интерфејс не мења. Преводилац се може мењати од енглеског на француски подразумевајући да се оба слажу и да ни један не мења интерфејс ка слоју 1 или слоју 3.



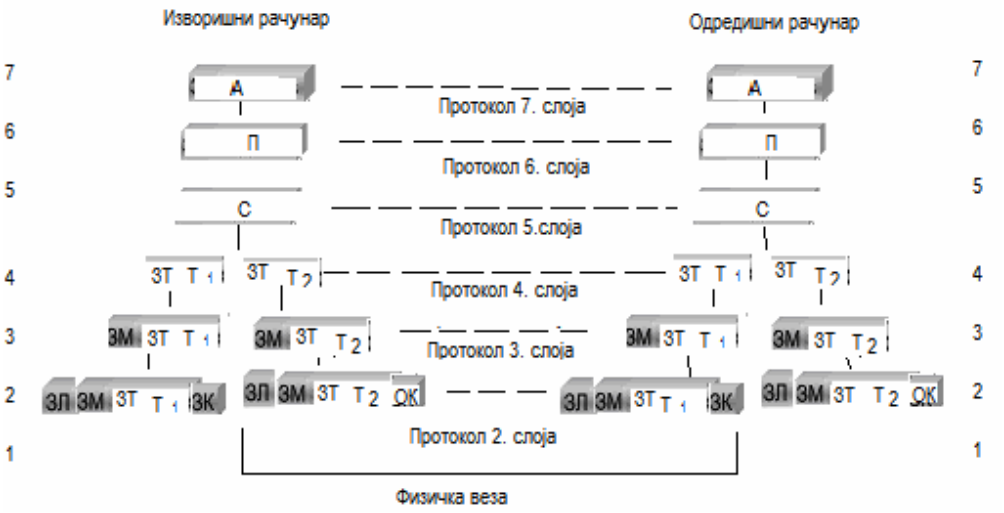
Слика 1.1 Пример комуникације истраживача – парњака

Посматрајмо технички пример: како обезбедити виртуелну везу ка највишем слоју седмослојне мреже. На слици 1.2 поруку А генерисао је процес који се одвија на 7. слоју. Порука се заједно са заглављем прослеђује са 7. до 6. слоја преко интерфејса 6/7 слоја.

У овом примеру 6. слој трансформише поруку, на пример врши компресију текста, а затим прослеђује нову поруку П ка 5. слоју преко интерфејса 5/6 слоја.

Слој 5, у овом примеру, не модификује поруку, већ једноставно регулише смер преноса (нпр. спречава долазеће поруке да дођу у 6. слој док је он заузет и води рачуна о серији одлазећих порука ка 5. слоју.

У нашем примеру, као и у другим стварним мрежама, не постоји ограничење у величини поруке коју може 4. слој да прими, али постоји ограничење које поставља 3. слој. Посматрајући даље, 4. слој мора да разбије долазећу поруку Т у мање јединице, додајући заглавље (ЗТ) свакој јединици. Ово заглавље укључује управљачке информације, као што је нпр. редни број[1], да би омогућио 4. слоју на одредишном рачунару да повеже делове у правилном редоследу.



Слика 1.2 Пример виртуелне везе

На 3. слоју свакој поруци се додаје заглавље (ЗМ) и прослеђује 2. слоју. Други слој поред заглавља (ЗВ) додаје и ознаку о крају поруке (ОК). Тако формиран рам прослеђује се физичком везом.

Важна ствар за разумевање слике 1.2 је веза између виртуелне и стварне комуникације и разлика између протокола и интерфејса. Процес парњак[2] на 7. слоју, на пример, размишља о својој комуникацији као да је „хоризонтална", користећи протокол 7. слоја. Сваки од њих има процедуру која се назива „пошаљи на другу страну" и процедуру „узми са друге стране", иако ове процедуре стварно комуницирају са нижим слојевима преко интерфејса 7/6 слоја, а не са другом страном.

Апстракција са процесима парњака је суштинска за пројектовање мрежа. Без ове технике не би било могуће раздвојити пројектовање комплетне мреже (нерешиви проблем) у неколико мањих целина, које се могу надгледати, пројектовати и које се називају пројектовање појединих слојева.

Циљеви у пројектовању слојева

Сваки слој мора да има механизам за успостављање везе. Пошто се мрежа нормално састоји од много рачунара, а неки од њих садрже више процеса, потребан је начин да се

[1] Користи се и термин секвенцијски број (*sequence number*).
[2] *Peer process*

процесу на једном рачунару укаже са ким он то жели да комуницира. Адресирање је потребно у било ком слоју где постоји више одредишта.

Као што постоји механизам за успостављање везе кроз мрежу, тако постоји и механизам за раскидање везе, када она више није потребна.

Други скуп одлука у пројектовању су правила за пренос података. Подаци могу да се преносе:

- само у једном смеру, што се назива симплекс (једносмерна) комуникација,
- у оба смера (али не истовремено), што се назива полудуплекс комуникација, или
- у оба смера истовремено, што се назива потпуна дуплекс (истовремена двосмерна) комуникација.

Протокол мора, такође, да одреди колико логичких канала вези припада и какви су приоритети. Многе мреже обезбеђују најмање два логичка канала по једној вези, на пример: један за нормалне податке а један за брзе податке, један за корисничке а један за управљачке податке, један за један а други за други смер.

Контрола грешака важна је компонента када комуникациони путеви нису савршени. Познати су многи кодови за детекцију и корекцију грешака, али оба краја везе морају се договорити који се користи. Такође, прималац мора да има начин да укаже пошиљаоцу које поруке су коректно примљене а које нису.

Није тачно да баш сви комуникациони канали задржавају редослед порука које се шаљу. Да би се могао разрешити могући губитак редоследа, протокол мора да направи експлицитну припрему пријемнику да би омогућио да се делови могу исправно повезати. Видљиво решење је пребројавање делова, али ово решење још увек оставља отвореним питање шта треба урадити са деловима који стижу ван редоследа.

Питање које се појављује на сваком слоју је како спречити брзог пошиљаоца да преплави подацима спорог примаоца. Различита решења постоје. Сва укључују неку врсту повратне спреге од пријемника ка предајнику, директно или индиректно, зависи од тога каква је текућа ситуација пријемника.

Други проблем који се мора разрешити на различитим нивоима је способност процеса да прихвати произвољно дугачке поруке. Ова карактеристика доводи нас до механизма разлагања (сегментација), слања и поновног обнављања (асемблирање) поруке. Проблем који се јавља је и шта радити када процес инсистира на слању јединица података које су тако мале да је слање сваке понаособ неефикасно. Овде је решење скупити више малих порука које се шаљу ка заједничком одредишту у једну велику поруку, и раздвојити велику поруку на другом крају.

Када није згодно, или је скупо успостављање одвојених путева за сваки пар комуникационог процеса, слој испод може да одлучи да користи исти пут за више различитих веза. Ово мултиплексирање и демултиплексирање, уколико се ради транспарентно, може да користи било који слој. Мултиплексирање је потребно 1. слоју, на

пример, где се сав саобраћај за све везе мора послати преко једног, или у најбољем случају неколико физичких канала.

Када постоји неколико могућих путева између извориште и одредишта на истој тачки хијерархије, одлука о рутирању мора се донети. На пример, при слању података из Лондона у Москву одлучивање на вишем нивоу упутиће их преко Француске или Немачке, у зависности њихових закона, а одлучивање на нижем нивоу помоћи ће да се одабере расположиви канал у зависности од текућег саобраћаја.

Интерфејс и услуга

Функција сваког слоја је да обезбеди услугу слоју изнад себе. У овом делу анализираћемо појам услуге.

Активни елеменат у сваком слоју обично се назива целина[1]. Целина може бити софтвер (нпр. процес) или хардвер (као што је интелигентни улазно/излазни чип). Целине на истом слоју, на различитим машинама називају се парњак целине[2]. Целина у слоју N имплементира услугу који користи слој N+1. У овом случају слој N назива се давалац[3] услуге, а слој N+1 назива се корисник услуге. Слој N може да користи слој N-1 да би обезбедио себи услугу. Могуће је обезбедити више класа услуга нпр. брза, скупа комуникација, или спора, јефтина.

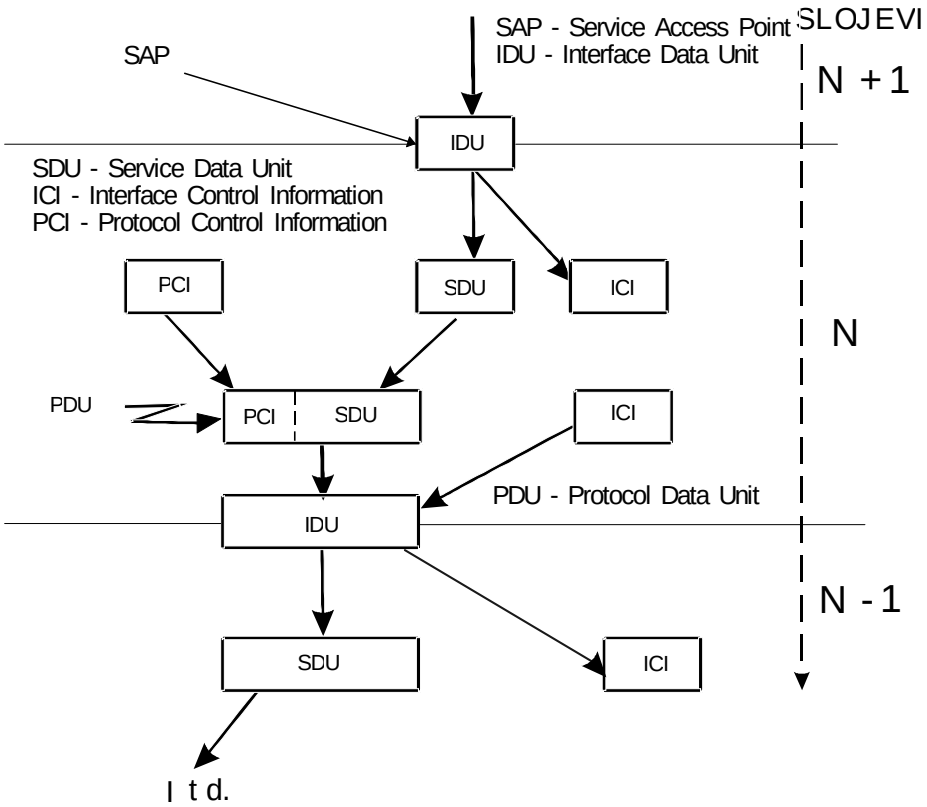
Услуга се добија на тачкама приступа услузи које се означавају са SAP[4]. Тачка приступа услузи N-тог слоја је место где слој N+1 може да приступи понуђеној услузи. Свака тачка приступа има адресу која је једнозначно идентификује. Да бисмо ово разјаснили тачке приступа у телефонском систему су утичнице у које се телефони могу прикључити, а адреса тачке приступа је телефонски број утичнице. Да бисте некога позвали морате знати његову адресу тачке приступа. У поштанском систему, адреса тачке приступа је улична адреса или поштански фах. Када треба послати писмо мора се знати адресу онога коме је писмо упућено.

Да би два слоја размењивала информације мора да постоји усаглашени договор око скупа правила везаних за интерфејс. Интерфејс целине слоја N+1 преноси јединице података интерфејса (које означавамо као IDU[5]) ка N-том слоју преко тачка приступа (SAP), као што је приказано на слици 1.3. Јединица података интерфејса (IDU) састоји се од јединице података услуге (SDU[6]) и управљачких информација. Јединица података услуге је информација која се прослеђује преко мреже ка парњак целини и затим ка N+1 слоју.

Управљачке информације потребне су нижим слојевима да би обавиле функцију као што је нумерисање бајтова у јединица података услуге али нису део самих података. Да би

[1] *Entity*
[2] *Peer entities*
[3] *Provider*
[4] *Service Access Points*
[5] *Interface Data Unit*
[6] *Service Data Unit*

пренела јединицу података услуге, целина слоја може да је дели у неколико делова. Сваки од тих делова може да има своје заглавље и да буде послат као посебна јединица података протокола (PDU[1]). Заглавље јединица података протокола користе парњак целине да би реализовале свој парњак протокол. Он идентификују која јединица протокола садржи податке, која садржи управљачке информације, која садржи редне бројеве, итд.



Слика 1.3 Пренос јединица података између слојева

Услуге са и без успоставе везе

Слојеви нуде два различита типа услуга ка слоју изнад њих: са успоставом везе[2] и без успоставе везе[3].

Услуга са успоставом везе је моделиран по узору на телефонски систем у коме да бисте са неким разговарали подижете слушалицу, окрећете број, разговарате и и на крају разговора спуштате слушалицу. Слична је употреба услуге са успоставом везе: корисник

[1] *Protocol Data Unit*
[2] *Connection Oriented*
[3] *Connectionless*

услуге прво успоставља везу, користи везу и раскида је. Суштински веза функционише као цев: пошиљалац шаље објекте (битове) на једном крају, а пријемник их преузима у истом редоследу на другом крају.

Услуга без успоставе везе моделирана је по узору на поштански услугу. Свака порука (писмо) садржи комплетну адресу и свака је упућена (рутирана) независно од осталих. Нормално, када се две поруке шаљу на исто одредиште, прва која се пошаље прва ће и стићи. Може се десити да прва порука која се пошаље закасни тако да друга стигне пре ње. Са системом са успоставом везе ово не може да се деси.

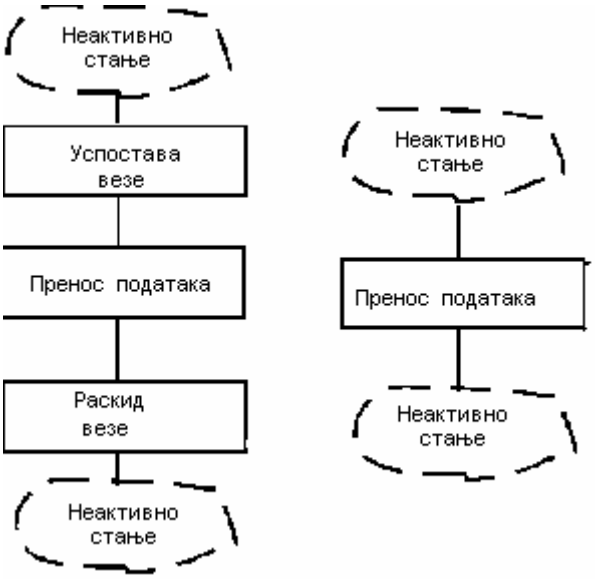
Свака услуга може се окарактерисати преко квалитета услуге (QoS[1]). Неке услуге су поуздане у смислу да је вероватноћа губитка података занемарљива. Обично се, за поуздану услугу користи потврда, тако да је пошиљалац сигуран да је порука стигла на одредиште. Процес потврде уводи додатне податке (потврду) и додатно кашњење, које је некада вредно тога а некада је непожељно.

Пренос датотека је типична ситуација где је поуздана услуга са успоставом везе неопходна. Власник датотеке жели да буде сигуран да су сви битови тачно примљени и у истом редоследу у коме су и послати. Мали је број корисника који би пристао да дође до мешања или губитка само пар битова, без обзира што би процес преноса био бржи.

У неким применам кашњења која се уносе при слању потврде су неприхватљива. Једна таква апликација је пакетизовани говор. За корисника телефона боље је да има шум са линије него уношење кашњења које се добија користећи потврде. Исто је и са преносом пакетизоване живе слике.

Постоје апликације којима није потребна успостава веза. На пример електронска пошта. Потребно је послати поруку и велика је вероватноћа да ће она бити примљена, али се не гарантује. Пакетска порука којом се шаље целокупна информација са адресама пошиљаоца и примаоца назива се датаграм.

[1] *Quality of Service*



Слика 1.4 Карактеристике услуга са успоставом и без успоставе везе

Примитиве услуге

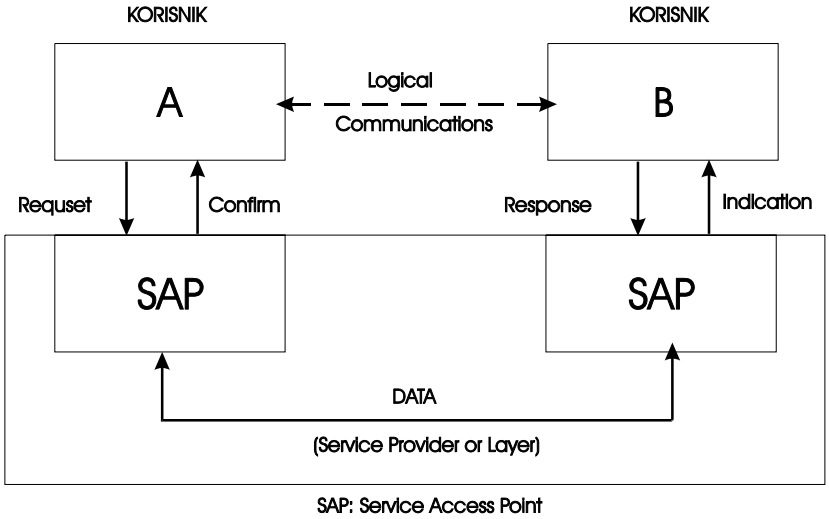
Услуге се формално специфицира преко скупа примитива (операција) које су доступне кориснику или другој некој целини и преко којих он приступа услузи. Ове примитиве налажу целини која пружа услугу да изврши неку активност, или да извести о некој активности коју су парњак целине обавиле. Један од начина да се класификују примитиве услуге је да их групишемо у четири класе, као што је приказано у табели 1.1.

Примитиве	Значење
Request - Захтев	Корисник тражи од целине која нуди услугу да одради неки посао
Indication - Индикација	Корисник је информисана о догађају
Response - Одговор	Корисник жели да одговори на догађај
Confirm - Потврда	Одговор на претходни захтев је стигао натраг

Табела 1.1 Четири класе примитива услуга

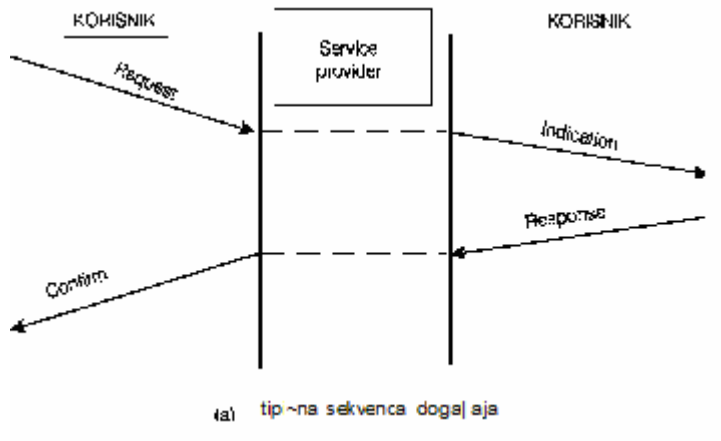
Да бисмо илустровали употребу примитива, размотримо како се веза успоставља и раскида. Целина која је иницијатор извршава примитиву *Request*, која резултује слањем јединице података протокола. Пријемник добија примитиву *Indication* (обавештење) да нека целина негде жели да успостави са њим везу. Користи *Response* примитиву која

указује да ли ће да прихвати или одбаци позив. Целина која је иницирала *Request* примитиву проналази шта се дешава посредством *Confirm* примитиве (слике. 1.5 и 1.6).



Слика 1.5 Успостављање логичке везе

Примитиве могу да имају параметре и многе од њих и имају. Параметри за *Request* примитиву могу да специфицирају рачунар са којом треба успоставити везу, тип услуге који се жели и максимална величина поруке која се у вези користи.



Слика 1.6 Размена примитива

Параметар за *Indication* може да садржи идентитет позивајуће целине, тип услуге који се жели и препоручену максималну величину поруке. Уколико се позвана целина не слаже са препорученом максималном величином поруке, може да направи другачији предлог у *Response* примитиви, која ће бити доступна пошиљаоцу преко примитиве *Confirm*. Детаљи договора су део протокола.

Релација између услуге и протокола

Услуге и протоколи имају различите концепте, који се често мешају. Услуга је скуп примитива (операција) које слој обезбеђује слоју изнад себе. Услуга дефинише које операције је слој спреман да изврши на захтев корисника, али не указује ничим како су ове операције примењене. Услуга је везана за интерфејс између два слоја, са нижим слојем који је давалац услуге и вишим слојем који је корисник услуге.

Протокол, на другој страни, је скуп правила која одређују формат и значење рамова, пакета или порука које се размењују преко парњак целина у оквиру слоја. Целине користе протокол у циљу имплементације дефинисане услуге. Могу слободно да мењају протокол али не могу да мењају услугу која је видљива њиховом кориснику. На овај начин услуга и протоколи потпуно су раздвојени.

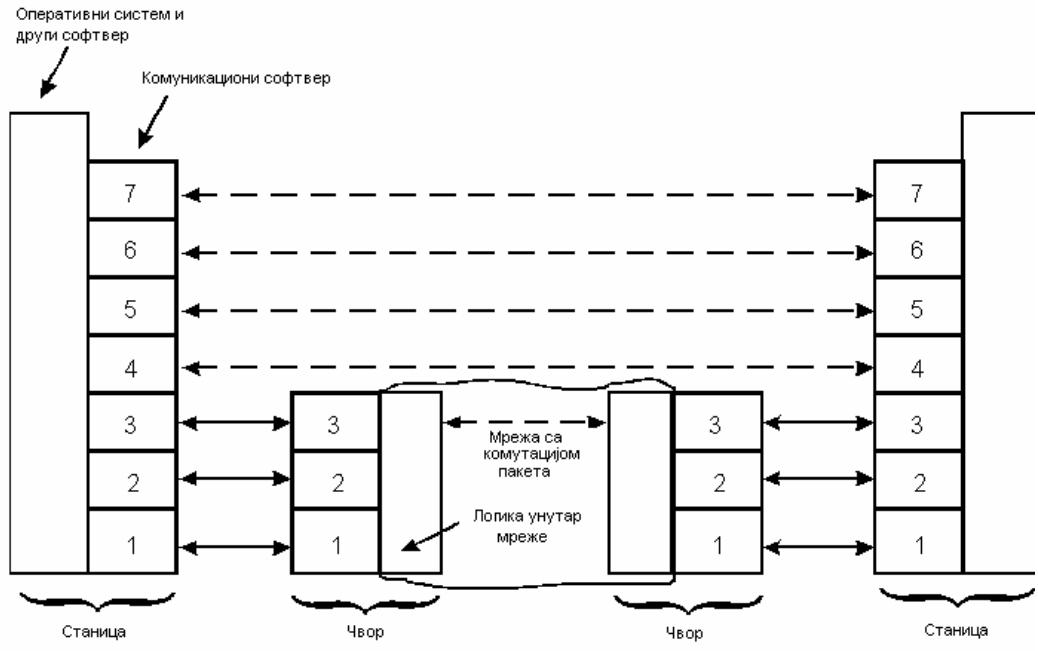
Направићемо аналогију са језиком за програмирање. Услуга је као апстрактни тип података или објекат у објектно - оријентисаним језицима. Он дефинише операције које треба да се изврше над објектима, али не специфицира како су ове операције имплементиране. Протокол се односи на имплементацију услуге и као таквог га корисник услуге не види.

Пошто смо размотрили слојевитост на апстрактан начин, у овом поглављу. Анализираћемо три важне мрежне архитектуре, OSI, X.25 и TCP/IP референтне моделе.

1.1 OSI референтни модел

OSI модел (слика 1.8) је базиран на препорукама које је развила интернационална организација ISO. Модел се назива ISO OSI референти модел за међусобну комуникацију отворених система[1] зато што се бави повезивањем отворених система – система који отворено комуницирају са другим системима.

[1] *Open System Interconnection reference model*



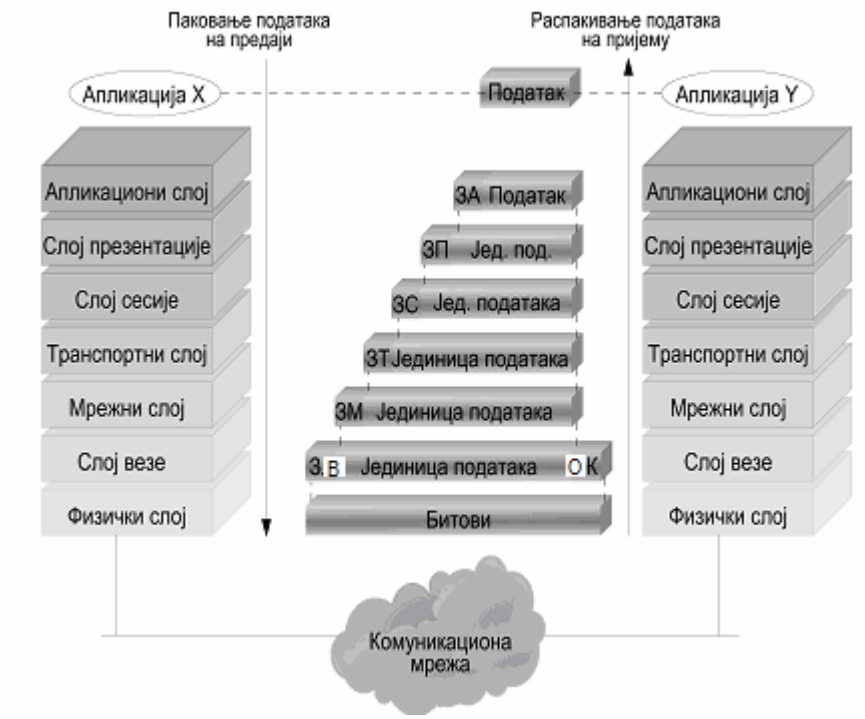
Слика 1.8 OSI референтни модел

Процес укалупљивања

Слика 1.9 приказује OSI референти модел и употребу јединице података у оквиру OSI референтног модела. Када апликација X има податке за апликацију Y она их шаље целини (процесу) на апликационом слоју. Подацима се додаје заглавље ЗА (тзв. процес укалупљивања[1]) које садржи управљачке информације потребне парњаку, 7. слоју на одредиште. Оригинални подаци заједно са заглављем се сада прослеђују 6. слоју. Презентациони слој посматра јединицу података као јединствену целину и додаје своје заглавље ЗП (друго укалупљивање).

Овај процес се наставља све до слоја везе (линка) који додаје и ознаку о почетку (ЗВ) и крају[2] (ОК) јединици података. Јединица података 2. слоја, која се назива рам (оквир) преко физичког слоја прослеђује се на трансмисиони медијум. Када рам стигне до одредишног система на њему се одвија процес обрнутог редоследа. Сваки слој примивши јединицу података уклања управљачке информације (заглавље) њему упућене а осатак прослеђује вишем слоју.

[1] Користи се и термин учауривања (*encapsulation*).
[2] *Trailer*



Слика 1.9 Ток података (укалупљивање)

Физички слој

Физички слој дефинише физички интерфејс између уређаја и правила по којима се сигнали преносе од једног до другог. Физички слој има четири важне карактеристике:

- механичку,
- електричну,
- функционалну,
- процедуралну.

Пример стандарда за физички слој је EIA[1] 232-F.

Слој везе

Док физички слој обезбеђује само услугу која се састоји у формирању сигнала (нпр. као серије битова), задатак слоја везе[2] је да обезбеди да веза буде и поуздана. Слој везе обезбеђује и начин да се успостави, одржава и раскине веза. Основна услуга, коју обезбеђује слој везе вишим слојевима, је детекција грешке и управљање везом. Тако, са потпуно функционалним протоколом слоја везе, следећи виши слој може претпоставити виртуелни пренос без грешака. У ствари, уколико је комуника ција између два система

[1] Раније означен као RS-232.
[2] *Data Link*

који нису директно повезани, веза између тих система састоји се од већег броја физичких веза (линкова), при чему свака физичка веза функционише независно. На тај начин виши слојеви нису ослобођени одговорности за контролу грешке.

Мрежни слој

Основна услуга мрежног слоја је да обезбеди механизме за транспарентан пренос података између целина транспортног слоја (транспортних целина). Ослобађа транспортни слој потребе да зна било шта о преносу података или техникама комутације које се користе да би се повезали системи. Услуга мрежног слоја (мрежна услуга) је одговоран за успостављање, одржавање и раскидање везе преко комуникационих путева који су у ту везу укључени.

То је тачка у којој је концепт протокола помало нејасан. Ово је најбоље илустровано на слици 1.8 која показује две станице које међусобно комуницирају не преко директног линка већ преко мреже са комутацијом пакета. Станице имају директне линкове ка мрежним чворовима. Протоколи 1. и 2. слоја су протоколи станице-чвора (локални). Јасно је да су протоколи 4. до 7. слоја протоколи између 4. до 7. целине двеју станица. Протокол 3. слој је помало од оба.

Принципијелни дијалог одвија се између две станице и чвора; станица шаље адресиране пакете ка чвору за прослеђивање ка мрежи. Чвор захтева комуникацију преко виртуелног кола, користи везу да би пренео податке, и завршава везу. Ово је све урађено протоколом станица-чвор. У ствари, пошто се пакети размењују и виртуелни канал се успоставља између две станице, постоји такође и аспект протокола станица-станица.

Постоји широки спектар комуникационих карактеристика које се јављају, а које може да подржава мрежни слој. На једном крају најједноставнија је директна веза између станица. У овом случају, може да постоји мало, или нимало потребе за мрежним слојем, пошто слој везе може да извршава неопходне функције за одржавање везе између крајњих тачака везе.

Уобичајена услуга 3. слоја је да води рачуна о детаљима коришћења комуникационе мреже. У овом случају мрежна целина у станици мора да обезбеди мрежу потребним информацијама о комутирању и рутирању података ка другој станици.

Транспортни слој

Слој 4 и слојеви изнад у OSI моделу генерално се називају виши слојеви. Протоколи на овим слојевима су с краја - на крај[1] и не воде рачуна о мрежи. Намена 4. слоја је да обезбеди поуздан механизам размене података између процеса на различитим системима.Транспортни слој гарантује да се јединице података прослеђују без грешака, по правилном редоследу, без губитка или удвостручавања. Транспортни слој може такође да води рачуна о оптимизацији коришћења услуге мрежног слоја и обезбеђивању захтеваног квалитета услуге за целине слоја сесије. На пример, целина сесије може да специфицира прихватљиву вероватноћу грешке, максимално кашњење, приоритет и

[1] *End to end*

поузданост. Практично, транспортни слој служи као корисничка веза са комуникационим уређајима.

Величина и комплексност транспортног протокола зависе од типа услуге кој(у)и може да добије од 3. слоја. За поуздан рад 3. слоја, са виртуелним каналом, потребне су минималне функције 4. слоја. Уколико је 3. слој непоуздан, или само подржава рада са датаграмима[1], протокол 4. слоја укључиће механизме за детекцију и опоравак од грешке. ISO је дефинисао четири класе транспортног протокола. Свака обезбеђује различите услуге.

Слој сесије

Слој сесије обезбеђује механизам за дијалог између апликација. Минимално, слој сесије обезбеђује начин на који ће два апликациона процеса да успоставе и користе везу која се назива сесија. Поред тога он може да обезбеди следеће услуге:

- тип дијалога који може да буде двострано симултан, двострано наизменичан или једносмеран,
- опоравак од грешке је последица тога што слој сесије може да обезбеди механизам за проверу. Уколико се неки тип грешке појави између контролних тачака слој сесије може поново да пошаље све податке од последње контролне тачке.

Презентациони слој

Слој презентације (презентациони слој) бави се синтаксом података који се размењују између апликационих целина. Његова намена је да разреши разлике у формату и репрезентацији података. Презентациони слој дефинише синтаксу која се користи између апликационих целина и обезбеђује одабир даљих модификација репрезентација које ће се користити.

Пример презентационог протокола је телетекст, видеотекст, шифрирање и протокол виртуелног терминала.

Апликациони слој

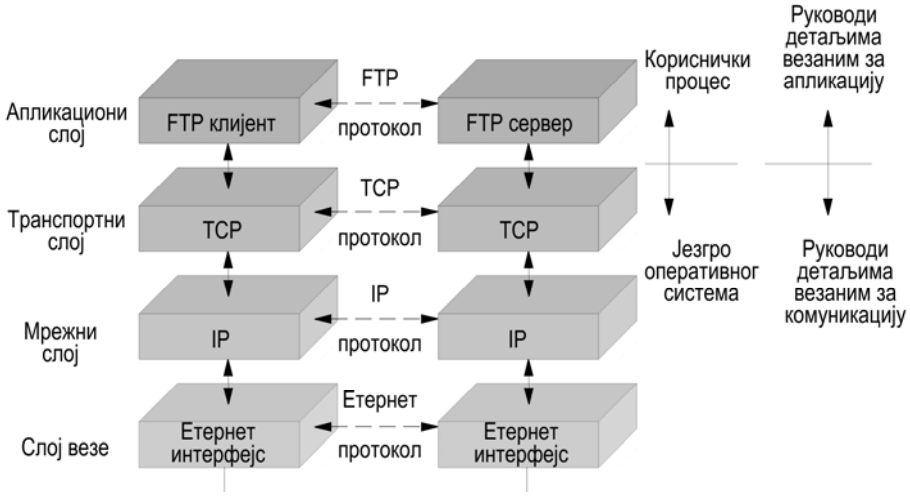
Слој апликације (апликациони слој) обезбеђује апликационом процесу приступ OSI окружењу. Овај слој садржи функције управљања и генерално корисне механизме за подршку дистрибуираним апликацијама. Пример протокола на овом слоју је протокол пренос, приступ и управљање датотекама[2].

1.2 TCP/IP референтни модел

TCP/IP је четворослојни референтни модел чији је један пример приказан на слици 1.10. Развила га је 1960. године агенција DARPA за мрежу са комутацијом пакета. Постао је

[1] *Datagram* - јединица података протокола која садржи адресу изворишта и одредишта, податке и која се потпуно независно може преносити кроз рачунарску мрежу.
[2] *File transfer Access and Management*

начин за повезивање рачунара који се највише користи, тј. *de facto* стандард. TCP/IP протоколи су основа Интернета.



Слика 1.10 TCP/IP референтни модел

Скуп TCP/IP протокола је комбинација различитих протокола са различитих слојева. Сваки од слојева има различита задужења:

- Слој везе (среће се у литератури и под називом приступ мрежи) води рачуна о свим детаљима физичког медијума који се користи.
- Мрежни слој (среће се и под називом Интернет слој) води рачуна о кретању пакета кроз мрежу. Преусмеравање (рутирање) пакета извршава се на овом слоју. Протоколи IP, ICMP и IGMP чине мрежни слој у TCP/IP скупу протокола;
- Транспортни слој апликационом слоју изнад себе обезбеђује проток података између две станице (рачунара). У TCP/IP скупу протокола најважнија два протокола су:
  - TCP који обезбеђује поуздан проток података између рачунара, најчешће преко виртуелних веза. Од података које добија од апликације: формира целине (делове) које прослеђује мрежном слоју, шаље потврде о успешном пријему пакета од удаљених рачунара, поставља времена на часовницима, итд.,
  - UDP обезбеђује много једноставнију услугу апликационом слоју. Он једноставно шаље од једног до другог рачунара пакете података (датаграме). Све захтеве за поузданшћу мора да преузме апликациони слој.
- Апликациони слој води рачуна о карактеристичним детаљима сваке апликације. У скоро свим имплементацијама обезбеђене су протоколи за следеће апликације:

- Telnet – за удаљени приступ;
- FTP[1] - за пренос датотека;
- SMTP[2] - за електронску пошту;
- SNMP[3] –за надзор и управљање рачунарском мрежом.

1.3 ATM референтни модел

ATM је систем са комутацијом и преносом пакета који су тачно одређене величине од 53 бајта Назива се и систем са комутацијом ћелија.

Стандард је донела интернационална организација ITU-T а референтни модел се разликује и од OSI и од TCP/IP референтних модела. На слици 1.11 представљен је ATM референтни модел. Физички слој води рачуна о специфичностима трансмисионог медијума и техникама кодирања.

Два слоја референтног модела односе се на ATM функције: ATM слој заједнички за све услуге које омогућавају пренос и слој за прилагођавање на ATM слој (AAL[4]). ATM слој дефинише пренос података помоћу ћелија и дефинише начин коришћења виртуелних (логичких) веза.

Намена адаптационог слоја је да обезбеди да и други (не само ATM) протоколи користе ATM системе. AAP слој повезује информације виших слојева у ATM ћелије (на изворишној страни), усмерава пренос ћелије преко ATM мреже и на одредишној страни преузима информације из ATM ћелија и испоручује их вишим слојевима.

Референтни модел је тродимензионалан и сачињавају га следеће одвојене равни:

- корисничка раван 1 - обезбеђује кориснику све информације потребне за пренос података заједно са управљачким информацијама као што су контрола тока, контрола грешке итд,
- контролна раван 2 - извршава све операције везане за управљање позивом и везом,
- управљачка раван 3 и управљање слојевима 4 - воде рачуна о функцијама везаним за надзор ресурса и координацију између слојева.

[1] *File Transfer Protocol*
[2] *Simple Mail Transfer Protocol*
[3] *Simple Network Management Protocol*
[4] ATM *Adapatation Layer*

Слика 1.11 ATM референтни модел

1.4 X.25 референтни модел

Да би се спречило да свака од земаља прави међусобно некомпатибилне интерфејсе, интернационална организација ITU-T[1] донела је стандарде за протоколе за приступ мрежи са комутацијом пакета. Заједно су ови стандарди познати као X.25. Стандард X.25 дефинише (слика 1.12):

- интерфејс између опреме крајње станице[2] која се означава као опрема крајње станице DTE[3] и опрема носиоца (оператера) која се означава DCE[4]. Опрема чвора се означава са DSE[5],
- формат и значење инфорамција које се размењују преко DTE - DCE интерфејса и то за протоколе слојева 1, 2 и 3 (слика 1.12). Пошто овај интерфејс раздваја опрему носиоца (DCE) и опрему корисника (DTE) важно је да је интерфејс пажљиво дефинисан.
- три слоја комуникације: физички слој, слој везе и слој пакета.

Физички слој се бави физичким интерфејсом између крајњих станица (рачунара, терминала) које се прикључују на мрежу и физичке везе (линка) који повезује станице са пакетском мрежом. У реализацији интерфејса користе се спецификације за физички слој X.21 или EIA -232.

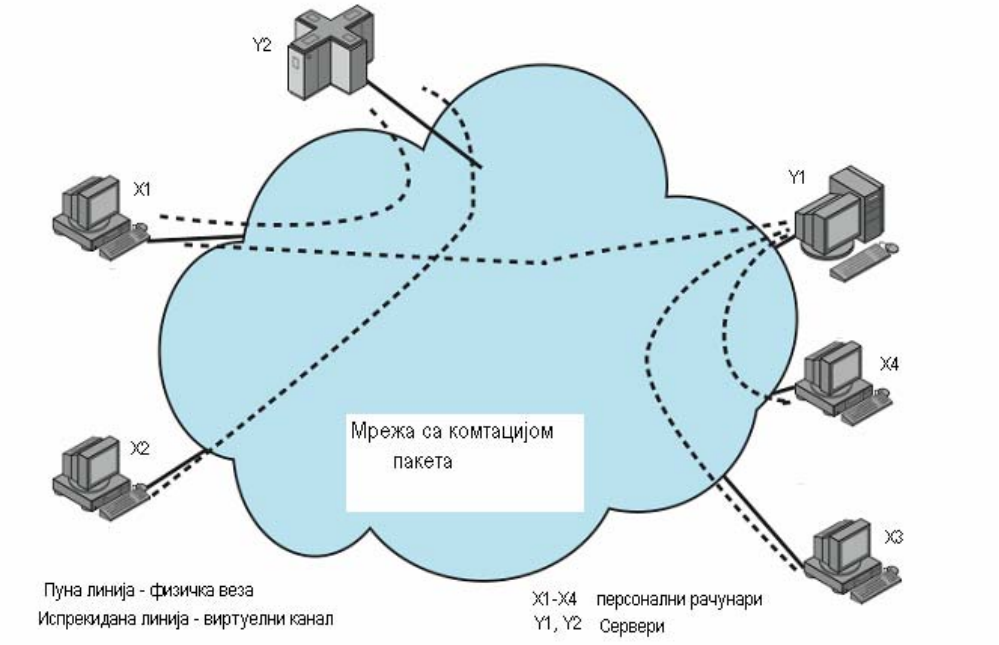
Слој везе обезбеђује поуздан пренос података преко физичког линка. На слоју везе користи се протокол за балансирани приступ (LAPB[6]) који је сличан HDLC протоколу[7].

[1] Некада CCITT.
[2] Користи се и термин хост (*host*).
[3] Опрема терминала за податке (*Data Terminal Equipment*).
[4] *Data Circuit Terminating Equipment*
[5] *Data Switching Exchange*
[6] *Link Access Protocol Balanced*
[7] Детаљније објашњење је дато у уџбенику Рачунарске мреже, 11. поглавље.

Пакетски слој обезбеђује услугу успостављања и одржавање логичке везе - виртуелног канала између два корисника.

Слика 1.12 Упоређивање X.25 и OSI референтних модела

У примеру на слици 1.13 станице (персонални рачунари) су успоставиле са серверима логичке везе и комуницирају преко одговарајућих виртуелних канала: станице X_1, X_3 и X_4 комуницирају са сервером Y_1, станице X_1 и X_2 комуницирају са сервером Y_2.

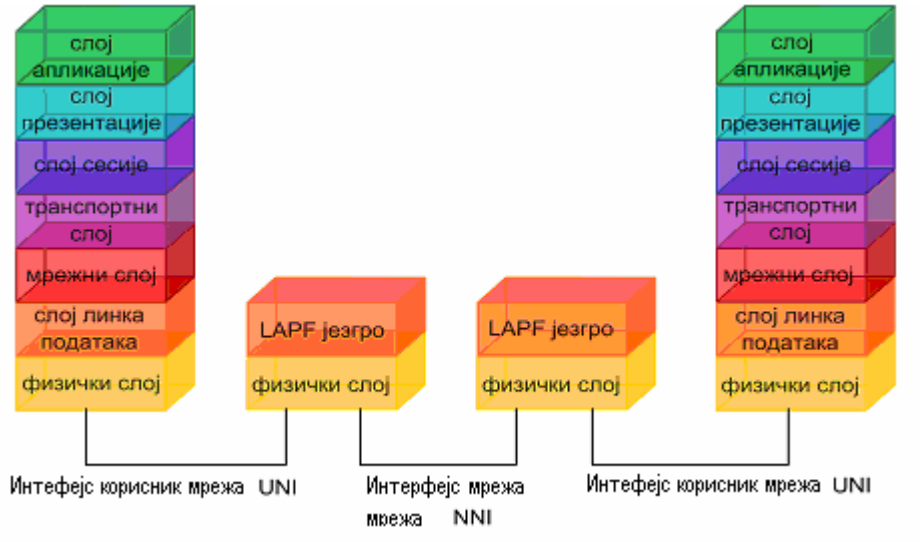


Слика 1.13 Начин употребе виртуелног канала

1.5 Штафетни пренос рамова

Данашњи преносни системи на великим растојањима[1] су највише ослоњени на оптичке каблове и значајно мање подложни сметњама него мреже пре двадесетак година. Као резултат тога примењен је нови концепт комутације - штафетни пренос рамова (FR[2]) или комутација рамова.

Основна идеја комутације рамова је следећа: корекције грешака и контрола протока се из чворова мреже, пребацују у крајње станице мреже (слика 1.14). Крајње станице као што су персонални рачунари, сервери, велики рачунари[3] користиће протоколе виших слојева (од 3. до 7.) и обезбедити сопствену контролу грешака. Овај концепт одлично функционише док су грешке у преносним системима мале. Уколико се овај, основни предуслов промени, онда долази до поновог слања података кроз целу мрежу уместо од једног до другог чвора (комутатора).



Слика 1.14 Упоређивање FR и OSI референтних модела на интефејсима корисник-мрежа[4] и мрежа-мрежа[5]

Циљ мрежа са штафетним преносом рамова је да омогући ефикаснији пренос података од X.25 мрежа са комутацијом пакета. За X.25 мреже карактеристично је да:

- управљачки пакети који се користе за успоставу и раскидање виртуелних канала се преносе истим виртуелним каналима као и пакети података[6],
- мултиплексирање виртуелних канала се обавља на 3. слоју,

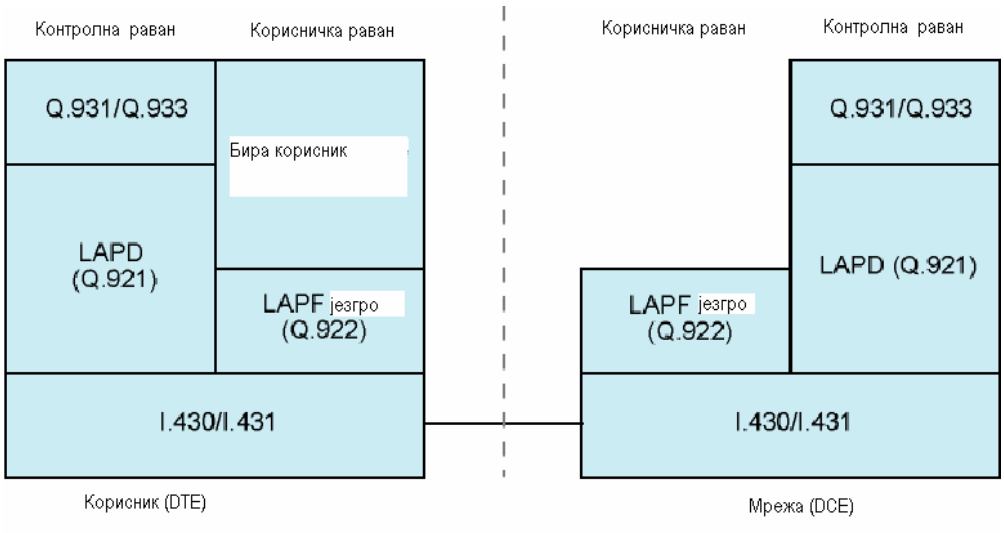
[1] *Long distance digital transmission system*
[2] *Frame Relay*
[3] *Mainframe* - мејнфрејм
[4] UNI (*User to Network Interface*)
[5] NNI (*Network to Network Interface*)
[6] *In band signaling*

- и 2. и 3. слој користе механизме за контролу тока и грешака.

Препорука I.233 (ITU-T) специфицира употребу штафетног преноса рамова за брзине приступа од 2Mb/s. Користе се и веће брзине приступа. На слици 1.16 је приказана слојевита архитектура протокола која се користи у мрежама са штафетним преносом рамова. Уочавају се две одвојене равни:

- контролна раван која се користи код успостављања и окончавања логичке везе
- корисничка раван која је одговорна за прено података између претплатника.

На тај начин протоколи контролне равни су између претплатника и мреже а протоколи корисничке равни су између крајњих корисника[1].



Слика 1.15 Станадарди у мрежи са штафетним преносом рамова

У табели 1.1 је дата намена стандарда означених на слици 1.15

Намена стадарда	ITU-T стандард	ANSI стандард
Архитектура и опис услуга	I.233	T1.606
Слој за податке	Q.922 (LAPF[2])	T1.618
Управљање перманентним виртуелним каналом (PVC[3])	Q.933	T1.617
Регулисање загушења	I.370	T1.606a
Сигнализација комутираним виртуелним каналом (SVC[4])	Q.933	T1.617

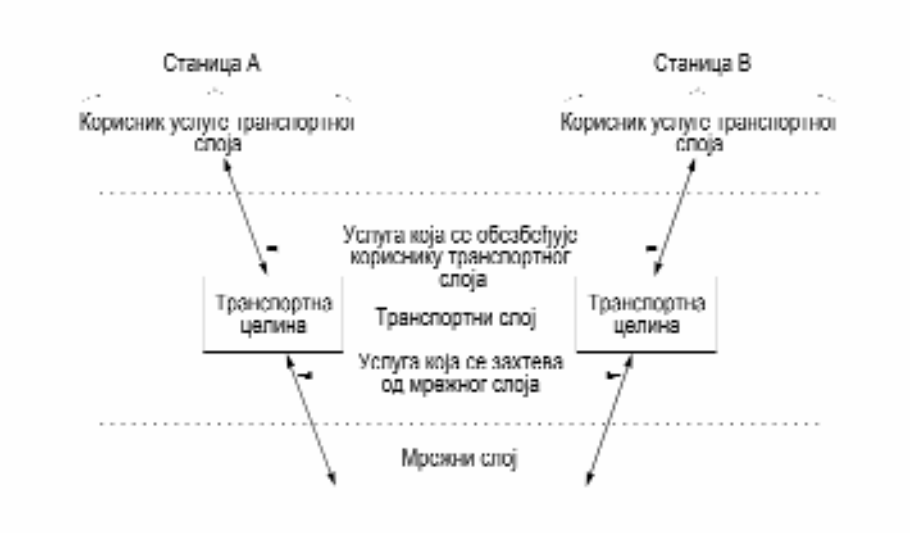
Табела 1.2 Ознаке и намене стандарда код мрежа са комутацијом рамова

[1] *End to end*
[2] *Link Access Protocol for Frame Mode Bearer Service*
[3] *Permanent Virtual Circuit*
[4] *Switched Virtual Circuit*

2. Транспортни слој

Транспортни слој је кључна карика у целокупном концепту архитектуре протокола. Протоколи нижих слојева су лакши за разумевање и мање сложени од транспортног протокола. Транспортни протокол обезбеђује услугу преноса података између крајњих корисника (с краја на крај). Свака апликација се може пројектовати тако да користи директно услугу транспортног слоја без посредства слоја сесије и презентације.

Најважнији циљ транспортног слоја је да обезбеди ефикасну, поуздану и економичну услугу својим корисницима, обично процесима на апликационом слоју[1]. Да би достигао овај циљ траснпортни слој користи услуге који му обезбеђује мрежни слој. Хардвер и/или софтвер у оквиру транспортног слоја који обавља те задатке назива се транспортна целина. Транспортна целина може бити реализована као део језгра оперативног система, као посебан кориснички процес у библиотеци повезаној са мрежном апликацијом или на мрежној интерфејсној картици. Повезаност (логичка) мрежног, транспортног и виших слојева приказана је на слици 2.1.



Слика 2. 1 Веза мрежног, транспортног и виших слојева

[1] Или слоју сесије ако је реч о OSI референтном моделу.

Услуга коју пружа транспорни слој је веома сличан услузи коју пружа мрежни слој[1]. Поставља се питање: ако је то тако зашто постоје и користе се два различита слоја? Зашто није довољан само један слој?

Разлог је једноставан. Софтвер који реализује услуге транспортног слоја налази се на рачунару самог корисника, док се софтвер који реализује услуге мрежног слоја најчешће налази на рутерима који су код рачунарских мрежа ширих географских подручја[2] део опреме оператера. Тиме је изнад мрежног слоја постављен још један слој - транспортни који може да побољша квалитет услуге мрежног слоја на месту и према захтеву крајњег корисника (апликације).

2.1 Услуге које пружа транспортни слој

Услуге које обезбеђује транспортни слој је пренос података с краја на крај на такав начин да штити кориснике од детаља везаних за комуникациони систем који користе. Категорије услуга које су корисне у описивању услуга траснпортно слоја су:

- тип услуге,
- степен услуге,
- пренос података,
- кориснички интерфејси,
- надзор везе,
- убрзана испорука података,
- извештај о статусу,
- безбедност.

Тип услуге

Могућа су два основна типа услуге: са успоставом везе[3] и без успоставе везе[4]. Услуга са успоставом везе обезбеђује успостављање, одржавање и раскидање логичке везе између транспортних корисника. Оваква услуга генерално је поуздана и садржи механизме као што су: управљање протоком, управљање грешкама и испорука у правилном редоследу.

Степен услуге

Протокол транспортне целине дозвољава кориснику да специфицира степен услуге или квалитет услуге који очекује. Транспортна целина ће покушати да оптимизира употребу мреже и везе. Примери услуге који се могу захтевати су:

---

[1] Користићемо термин транспортна услуга за услугу коју пружа транспортни слој и мрежна услуга за услугу коју пружа мрежни слој.
[2] WAN - *Wide Area Network*
[3] *Connection oriented*
[4] *Connectionless*

- прихватљив ниво погрешних и изгубљених података,
- пожељно и максимално прихватљиво кашњење,
- пожељна и минимална пропусност[1],
- нивои приоритета.

Јасно је да су могућности транспортне целине да обезбеди захтеване услуге ограничене могућностима услуга које нуде нижи слојеви. Навешћемо неке од апликација које захтевају специфични степен услуге:

- протокол за пренос датотека FTP[2] може да захтева велику пропусну моћ система и високу поузданост да би спречио поновно слање,
- трансакциони протоколи (упити база података) могу да захтевају мала кашњења,
- електронска пошта може да захтева више нивоа приоритета.

Постоје четири категорије транспортног протокола:

- поуздан протокол са успоставом везе,
- мање поуздан протокол без успоставе везе,
- протокол за пренос говора који захтева благовремени пренос и пренос исправног редоследа,
- протокол за пренос у реалном времену који захтева високу поузданост и минимална кашњења.

Пренос података

Намена транспортног протокола је да обезбеди пренос података између транспортних целина. Подаци и управљачке информације могу се преносити истим или различитим каналом. Могући је: потпуни дуплекс, полудуплекс и симплекс начин рада.

Интерфејс ка кориснику

Прецизно дефинисан интерфејс ка кориснику[3] нема потребе да буде стандардизован. Боље је да се у реализацији прилагоди специфичностима окружења коме се реализује. Веза између транспортне целине и корисника посматра се преко примитива (операција) и параметара који се размењују. Ове примитиве налажу целини да изврши неку активност, или да извести о некој активности коју су парњак целине обавиле. Један од начина да се класификују примитиве је да их групишемо у четири класе (табели 2.1). Примитиве које се користе су:

---

[1] *Throughput*
[2] *File Transfer Protocol*
[3] Користи се и термин кориснички интерфејс.

- захтев који иницира транспортни корисник да би покренуо одређену услугу,
- индикација коју обезбеђује транспортна целина да би обавестила о покретању активности за остваривањем одређене услуге,
- одговор који обезбеђује траспортни корисник као одзив на примитиву индикација,
- потврда која се враћа транспортном кориснику по реализацији тражене услуге.

На слици 2.2 приказан је временски редослед ових догађаја.

Слика 2.2 Временски редослед примитива (Поставити временску осу)

Примитиве	Значење
Request - Захтев	Захтева за одређеном услугом
Indication - Индикација	Индикација о покретању активности за реализацију услуге
Response - Одговор	Одговор на примитиву индикације
Confirm - Потврда	Потврда иницијатору о реализацији за одређене услуге

Табела 2.1 Четири класе примитива услуга

Надзор везе

Када је обезбеђена услуга са успоставом везе транспортна целина је одговорна за успостављање и раскидање (окончање) везе. Када је симетрична процедура обезбеђена корисници са оба краја везе могу да иницирају комуникацију. Асиметричне процедуре омогућавају реализацију симплекс веза.

Раскид везе може да буде тренутан или одложен. Када је тренутан раскид везе онда подаци који су послати, а још нису стигли на одредиште, могу бити изгубљени. Одложен

раскид спречава било коју страну да прекине везу док сви послати подаци не стигну на своје одредиште.

Убрзана испорука

Представља услугу која одговара механизму прекида и користи се за пренос повремено хитних података (нпр. аларм). Разликује се од услуге са приоритетом који додељује виши приоритет одређеним ресурсима. Подаци са тих ресурса се прослеђују брже од осталих података.

Извештај о стању

Извештај о стању и параметрима је услуга која даје могућност транспортном кориснику да обезбеди или да сам добије информације које се односе на услове транспортне целине или везе. Примери статусних информација су:

- карактеристике везе (нпр. пропусна моћ, средње кашњење...),
- адресе (мрежна, транспортна),
- класа протокола која је у употреби,
- текуће стање часовника,
- стање протокола (у контексту дијаграма стања),
- деградација у односу на захтевани степен услуге.

Сигурност

Транспортна целина може да обезбеди различите услуге везане за сигурност као што су:

- управљање приступом која се реализује на локалном и удаљеном крају,
- шифровање/ дешифровање података,
- рутирање преко поузданих веза и чворова уколико су на располагању.

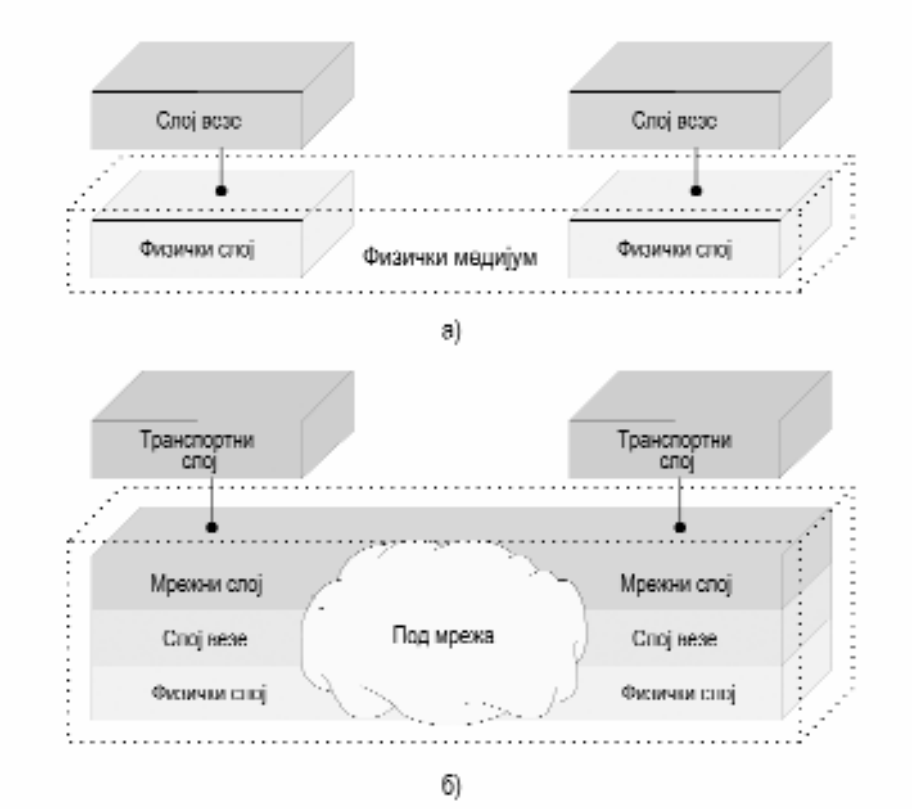
2.2 Елементи транспортног протокола

Транспортна услуга имеђу две транспортне целине реализује се помоћу транспортног протокола.Транспортни протокол сличан је протоколу слоја везе. Код оба протокола, поред осталог, имплементирани су механизми за контролу грешке, секвенционирање и контролу тока. Постоје и значајне разлике које ћемо објаснити у овом поглављу (слика 2.3).

Разлике које су на први поглед видљиве су следеће:

- два слоја везе комуницирају преко физичког канала док транспортни слојеви комуницирају преко подмрежа,

- успостављање везе преко физичког канала је једноставно, док је успостављање везе преко подмрежа знатно сложеније,
- подмрежа може да задржава податке и да их испоручи са значајним кашњењем,
- контрола тока и додељивање меморијског простора за чување пакета (бафероване) присутно је и код транспортног и код слоја везе. На слоју везе обично се резервише меморијски простор одређене величине за сваку од веза. Када рам стигне, меморијски простор за његово смештање је расположив. Пошто на транспортном слоју постоји велики број веза које треба опслужити принцип са наменским меморијским простором је практично неприхватљив.



Слика 2.3 Аналогија протокола: а) слоја везе б) транспортног слоја.

Сложеност транспортног протокола зависи у значајној мери од мрежне услуге. ISO је дефинисао три типа мрежне услуге:

- тип А за мрежне везе са прихватљивим степеном погрешних пакета и прихватљивим степеном сигнализираних грешака,

- тип B за мрежне везе са прихватљивим степеном погрешних пакета и неприхватљивим степеном сигнализираних грешака,
- тип C за мрежне везе са неприхватљивим степеном погрешних пакета.

Под погрешним пакетима подразумевају се изгубљени или дуплицирани пакети – (NPDU[1]). Уколико грешку уочи и исправи мрежна целина на такав начин да је невидљива за транспортног корисника проблем је успешно разрешен. Уколико мрежна целина детектује грешку, не може да је отклони него обавести (сигнализира) транспортну целину, онда се то назива сигнализирана грешка. У овом поглављу анализираћемо транспортни протокол за случај поуздане мрежног услуге и за случај непоуздане мрежне услуге.

2.3 Транспортни протокол и поуздана мрежна услуга

Претпоставка је да мрежна услуга прихвата поруке произвољне дужине и да их са вероватноћом од 100% испоручује у правилном редоследу. Ова претпоставка доприноси примени једноставног транспортног протокола. Примери комуникационих мрежа које обезбеђују овакву услугу су:

- X.25 мрежа са комутацијом пакета високе поузданости,
- мреже са штафетним преносом која користи LAPF[2] протокол,
- локалне рачунарске мреже по стандарду IEEE802.3 и LLC услугомом са успоставом везе.

Анализираћемо:

- адресирање,
- мултиплексирање,
- контролу тока и
- успоставу и раскидање везе.

Адресирање

Када апликациони процес (тј. корисник) жели да успостави везу са другим апликационим процесом потребно је да означи са којим то процесом, тј. потребно је да дефинише његову транспортну адресу. Термин који се користи[3] је тачка приступа транспортној услузи (TSAP[4]). На Интернету се ове крајње тачке -TSAP називају портови На мрежном слоју се ове тачке (адресе мрежног слоја) називају тачке приступа мрежној услузи

---

[1] *Network Protocol Data Unit* – јединице података мрежног слоја.

[2] *Link Access Procedure for Frame Mode Bearer Services*

[3] Дефинисан је у 1. поглављу

[4] *Transport Service Access Point*

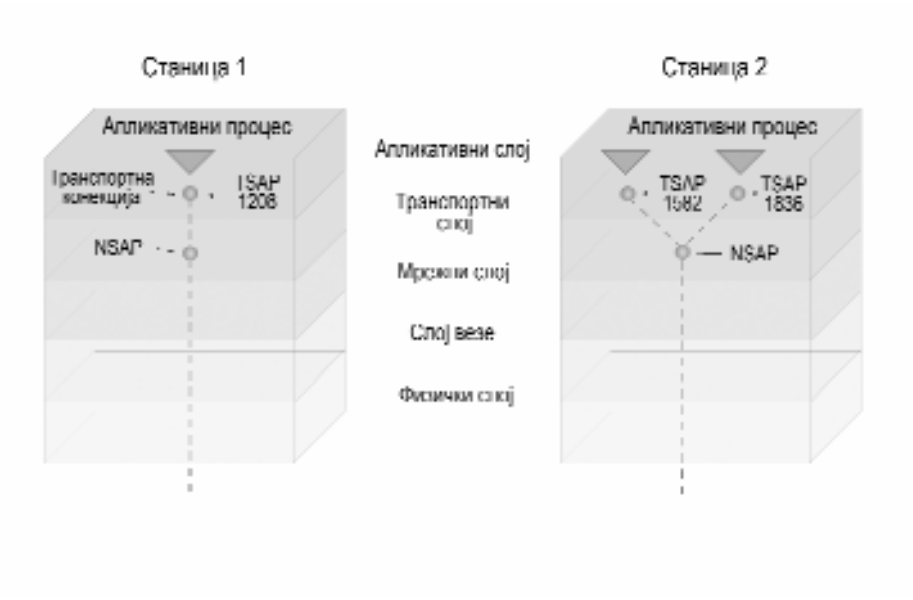
(NSAP[1]). Пример тачке приступа мрежној услузи (NSAP) су IP адресе. На слици 2.4 приказана је веза између тачака приступа транспортној и мрежној услузи и транспортне везе.

Апликациони процеси на страни клијента и сервера могу да се прикључе на тачку приступа транспортној услузи да би успоставили везу са удаљеном тачком приступа транспортној услузи. Ове везе се реализују преко тачака приступа мрежној услузи на страни сваке крајње станице (хоста).

Значи адреса корисника је одређена паром (станица, порт). Порт је променљив и представља одређеног корисника. Идентификација транспортне целине није потребна пошто обично постоји само једна целина за одређени тип протокола. Касније ћемо видети да адреса треба да укључи тип транспортног протокола (нпр. TCP, UDP).

У случају једне, одвојене, мреже станица представља уређај повезан на рачунарску мрежу. Када је реч о Интернету станица представља глобалну Интернет адресу.

Рутирање није у надлежности транспортног слоја, тако да он само прослеђује део адресе која се односи на станицу мрежној целини. Порт се налази у транспортном заглављу и на одредишту га може употребити одредишна транспортни целина.

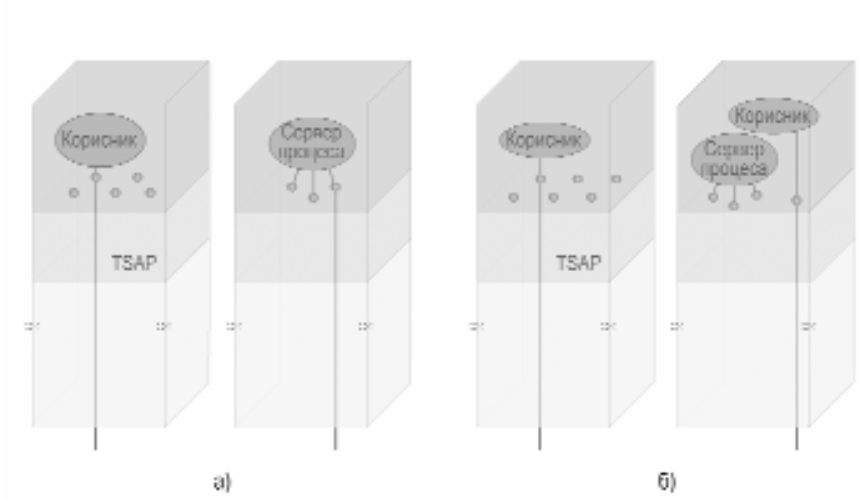


Слика 2.4 Транспортна веза, TSAP и NSAP

Поставља се питање како иницијатор транспорне везе зна адресу одредишног транспортног корисника? Одговор се може категоризовати на следећи начин:

[1] *Network Service Access Point*

- Корисник зна унапред одредишну адресу. У основи то је функција конфигурисања система.
- Неке од опште коришћених услуга проглашени су „познатим адресама”[1] (табела 2.2). На пример, то су серверске стране протокола FTP, SMTP и неки други стандардни протоколи у скуп TCP/IP протокола.
- Обезбеђени су сервери имена. Корисник захтева услугу користећи неко генеричко или глобално име. Захтев се шаље неком од сервера имена који садржи базу адреса. На основу имена проналази тражену адресу и шаље је кориснику. Добивши тражену адресу транспортна целина успоставља везу са одредиштем. Ова услуга је погодна за апликације које се често користе али повремено мењају своју локацију. На пример неки процес може се пребацити са једног рачунара на други у оквиру локалне рачунарске мреже да би се обезебедило равномерно оптерећење.
- У неким случајевима одредишни корисник је процес који се по захтеву треба покренути, односно изазвати[2]. Корисник иницијатор шаље захтев ка познатим адресама. Процес – сервер на тим адресама покреће нове процесе и шаље иницијатору адресу новопокренутог процеса. На слици 2.5а приказана је комуникација са сервером процеса, на сл. 2.5б комуникација са траженим корисником (изазваним процесом).



Слика 2.5 Како корисник успоставља везу са: а) сервером имена (процеса), б) покренутим процесом

[1] *Well-known address*. Листа се може наћи на *www.iana.org*
[2] *Spawned*

Порт	Протокол	Намена
21	FTP	Пренос фајлова
23	Telnet	Виртуелни терминал
25	SMTP	Електронска пошта
80	HTTP	*Word Wide Web*
110	POP-3	Удаљени приступ електронске поште
53	DNS	Сервер имена

Табела 2.2 Најчешће коришћени портови у скупу TCP/IP протокола

Мултиплексирање

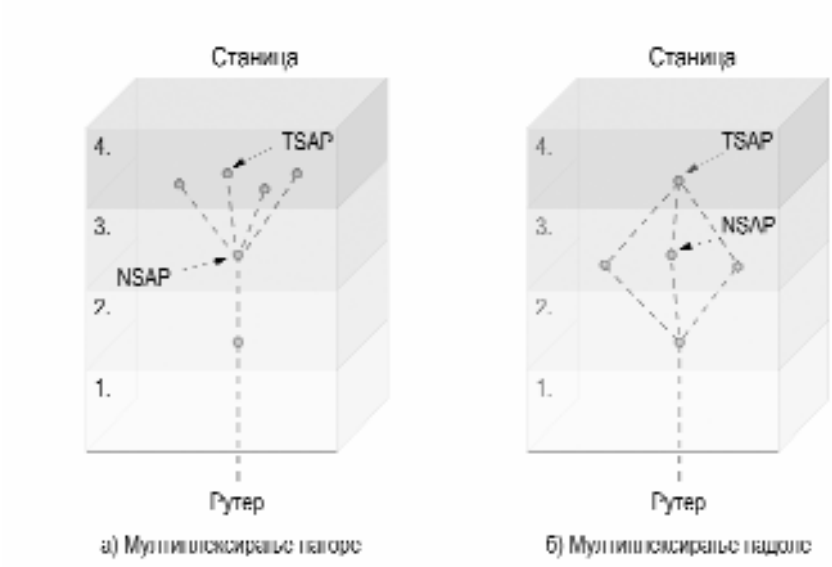
На транспортном слоју мултиплексирање је приказано на слици 2.6 и дефинишемо га као:

- мултиплексирања нагоре[1] уколико више транспортних корисника дели једну виртуелну везу успостављену између мрежних целина,
- мултиплексирање надоле[2] уколико једна транспортна веза користи више веза на мрежном слоју.

Навешћемо ситуације у којима се мултиплексирање користи. Узмимо за пример да је станици (хосту) на располагању само једна мрежна адреса тако да све транспортне везе на тој станици морају ту адресу да користе.

Претпоставимо да подмрежа користи виртуелне везе и намеће ограничење у максималној брзини података по једној вези. Уколико је кориснику потребна већа брзина од брзине једног виртуелног кола може да отвори више мрежних веза и подели саобраћај између њих. Уобичајени пример мултиплексирања надоле је коришћење две ISDN везе од по 64kb/s за повезивање са посредником Интернет услуга брзином од 128 kb/s.

[1] *Upward multiplexing*
[2] *Downward multiplexing*



Слика 2.6 Мултиплексирање нагоре и надоле

Контрола тока

Ово је једноставан механизам на слоју везе а сложен механизам на транспортном слоју из следећих разлога:

- контрола тока на транспортном слоју укључује међусобну интеракцију транспортних корисника, транспортних целина и услуга мрежног слоја,
- кашњење у преносу између транспортних целина може бити велико. То значи да постоји прилично кашњење у размени информација везаних за контролу тока,
- кашњење у преносу пакета између транспортних целина је у општем случају знатно дуже од времена потребног за његово слање,
- собзиром на то да транспортни слој функционише преко нижих слојева, кашњење може да буде веома променљиво. Ово доприноси томе да је прилично отежано коришћење часовника за поновно слање изгубљених података.

Основна сличност контроле тока на слоју везе (на пример код HDLC протокола), на мрежном слоју (на пример X.25 протокол) и на транспортном слоју (на пример TCP протокол) настаје из потребе да се спречи брзи пошиљалац да загуши спори пријемник. Код слоја везе пошиљалац чува рамове (баферује их) и на пријемној и на предајној страни. Када подмрежа обезбеђује непоуздану услугу транспортна целина - пошиљалац

чува све јединице података (TPDU[1]) док не добије потврду за њих. Пријемна транспортна целина може на пример да резервише меморијски простор[2] за све везе заједно а не за сваку појединачно.

Свака транспортна целина има одређену количину простора за чување пристиглих података, односно величину бафера. Јединице података (у даљем тексту сегменти[3]) који пристижу смештају се у бафер. Сваки сегмент се анализира нпр. TPDU заглавље и по завршетку анализе шаље се одређеном транспортном кориснику. Ако пријемна транспортна целина не може да одржи корак са дотоком сегмената, бафери бивају препуњени[4] и може да дође до губитка података. Да би се спречило појава „препуњених бафера", пријемна транспортна целина мора да предузме кораке за заустављње или успоравање дотока сегмената да би могла да обради већ пристигле сегменте. Овај задатак није једноставно реализовати с обзиром на могућу удаљеност (временску и просторну) транспортних целина. Изложићемо четири начина за разрешавање захтева који се постављају пред механизмом за контролу тока. Пријемна транспортна целина може:

- да не реагује ни на који начин,
- да одбије да надаље прима сегменте (TPDU) од мрежног слоја,
- да користи протокол клизајућег прозора са фиксним отвором прозора,
- да користи механизам кредита[5].

Прва варијанта значи да ће пријемна целина одбацити пакете у недостатку простора за њихово чување. Пошто не добије потврду за њих, пошиљалац их поново шаље. Овај приступ није пожељно користити, јер долази до ретрансмисије, а предност мреже са успоставом везе је та да не би требало да постоји ретрансмисија (или бар да се сведе на минимум). Друга варијанта је она која је примењена код протокола HDLC[6]. Кључни параметри су:

- свака јединица података добија редни број,
- отвор прозора је фиксан,
- користи се потврда која доводи до померања (клизања) прозора.

Са поузданим услугом мрежног слоја техника клизајућег прозора ради сасвим добро.

---

[1] *Transport Protocol Data Unit*
[2] *Buffer pool*
[3] Јединице података PDU (*Protocol Data Unit*) које међусобно размењују транспортне целине називају се сегменти.
[4] *Buffer overflow*
[5] Може се сматрати као променљиви отвор прозора.
[6] Детаљно је описано је у 11. поглављу [Рачунарске мреже].

У трећој варијанти пријемна транспортна целина поред потврде о пакетима које је успешно примила шаље нови параметар - кредит. Он представља отвор прозора али може да се мења у току трајање везе.

Успостава и завршетак везе

Да бисмо боље објаснили како се операције (примитиве) могу употребити у фазама успоставе и раскида везехпосматраћемо апликацију са сервером и већим бројем удаљених клијената. У табели 2.3 дат је преглед примитива и њихво значење.

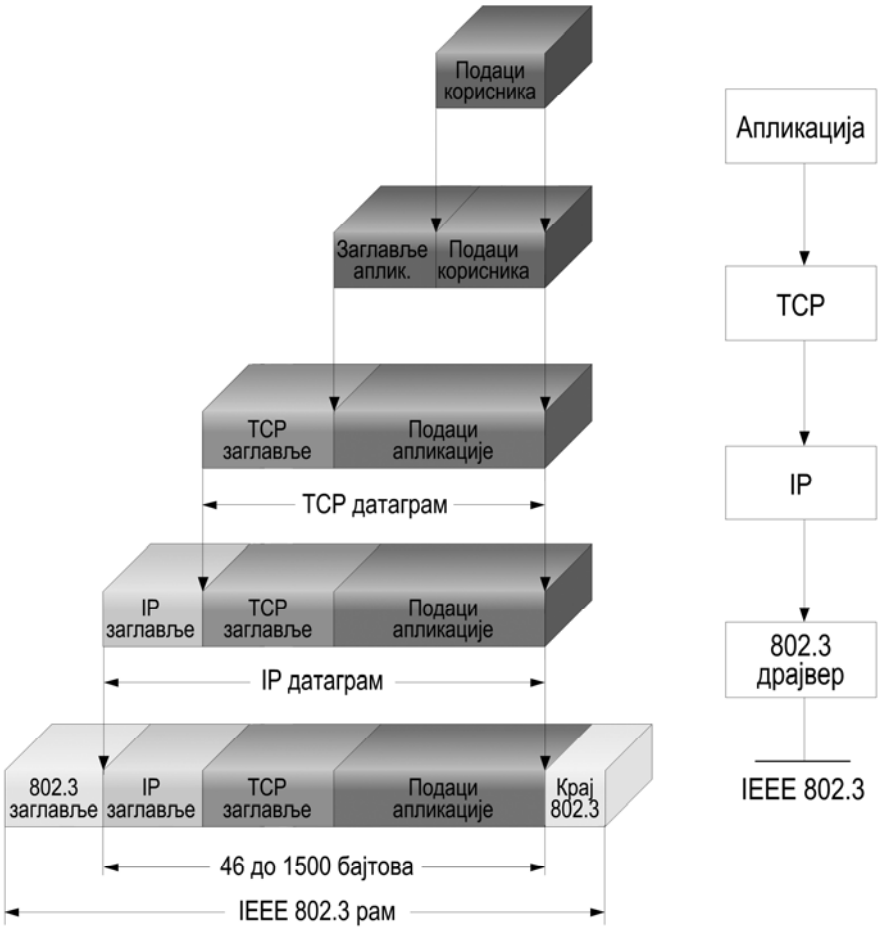
Примитива	Послата јединица података TPDU	Значење
Passive open[1]	Не шаље се ништа	Блокиран је док неки од процеса не покуша да се повеже са њим.
Active open[2]	Шаље захтев за успоставом везе	Активно покушава да успостави везу.
Send	Подаци	Шаљу се информације.
Receive	Не шаље се ништа	Блокиран је док не пристигну *Data* пакети.
Close	Шаље захтев за раскидом везе	Ова страна жели да оконча везу.

Табела 2.3 Примитиве и њихово значење

На почетку сервер извршава примитиву Passive open обично системским позивом који блокира сервер док се не појави захтев клијента.

Када клијент жели да комуницира са сервером извршава примитиву Active open. Транспортна целина извршава примитиву тако што блокира позивајућег корисника и пошаље серверу јединицу података „захтев за успоставом везе"[3]. Порука транспортног слоја клијента упућеног транспортном слоју сервера спакована (учаурена)[4] је у део за податке[5] пакета мрежног слоја. Пакет мрежног слоја спакован је у рам слоја везе. Када рам дође до сервера одвија се обрнут процес - распакивање на сваком слоју. Повезивање јединица података код TCP/IP референтног модела приказано је на слици 2.7.

[1] У литератури се пасивно отварање везе [Tanenbaum] означава и као Listen (ослушкуј).
[2] У литератури се активно отварање везе [Tanenbaum] означава и као Connect (повежи се).
[3] *Connection Request TPDU*
[4] *Encapsulated*
[5] *Payload*



Слика 2.7 Повезивање јединица података (PDU[1]) на транспортном, мрежном и слоју везе код TCP/IP референтног модела

За реализацију услуге са успоставом везе неопходне су процедуре за успоставу и завршетак везе. Три битне функције успоставе везе су:

- омогућавање сваком учеснику да се увери да је други учесник присутан,
- дозвољавање договора о опционим параметрима као што су на пример: максимална величина сегмента, максимална величина отвора прозора, квалитет услуге,

[1] *Protocol Data Unit*

- покретање додељивања ресурса потребних транспортној целини као што су меморијски простор за смештање података (бафер) или формирање табела.

Успостава везе је међусобни договор корисника и може се остварити разменом различитих команди и управљачким сегментима, као што је приказано на дијаграму стања на слици 2.9.

На почетку корисник је у неактивном стању - CLOSE[1], нема успостављену ни једну везу. Корисник командом (примитивом) Passive Open чека на захтев за успоставом везе. После издавања ове команде транспортна целина ствара (гради) објекат везе (нпр. улаз у табелу) која се налази у стању LISTEN[2]. Ако се транспортни корисник у међувремену предомислио и жели да се врати у CLOSE стање, може то једноставно да реализује извршавањем команде Close.

Уколико стигне сегмент *Syn (Cr)*[3] који је адресиран на корисника који је у стању пасивног чекања, веза је отворена и транспортна целина ће:

- сигнализирати кориснику да је веза отворена,
- послати *Ack Syn (Cc)*[4] пакет као потврду удаљеној транспортној целини,
- поставити објекат везе у стање ESTABLISHED.

Корисник може да успоставља (отвара) везу користећи команду Active Open[5] која покреће транспортну целину да пошаље сегмент *Syn*. Пријем одговарајућег *Syn* сегмента успоставља везу.

Када страна која је иницирала отварање везе прими *Syn* сегмент, може да пређе у ESTABLISHED стање. У случају да било који транспортни корисник изврши Close команду, веза се одмах прекида.

Било која страна може да иницира успостављање везе. У случају да обе стране иницирају везу у исто време, веза се без икаквих проблема успоствља. Разлог је то што се *Syn* сегменти посматрају и као захтеви за успоставом везе и као потврда о успостављеној вези. Веза се превремено прекида када локални корисник пошаље Close команду или када удаљена транпортна целина одбије везу слањем транспортне јединице података за окончање везе *Fin (Dr[6])*.

Шта се дешава уколико је корисник који би требало да се налази у стању пасивног чекања (LISTEN) неактиван (CLOSE)? Може да се деси:

---

[1] Указује да је веза затворена.
[2] Указује да је процес на страни сервера у стању ослушкивања позива клијента.
[3] TPDU захтев за успоставом везе *Cr (Connction reques*t). Код TCP протокола означава се са *Syn*.
[4] *Cc* (*Connection confirm*) - уобичајено је да се исти TPDU користи и као захтев за успоставом везе и као потврда о успостави везе и означава као *Ack Syn*. .
[5] Назива се и захтев за успоставом везе *(*Connect*)*.
[6] *Dr* (*Disconnect request*) - код TCP протокола означава се са *Fin*

- да транспортни слој одбаци захтев слањем *Fin*,
- да се захтев ставља у ред за чекање док корисник не пошаље команду Activ open,
- да транспортна целина сигнализира кориснику да постоји нерешен захтев за успоставом везе са њим.

У случају да транспортна целина реагује на трећи од наведених начина, Activ open команда није неопходна и може бити замењена командом Accept, која представља сигнал који шаље транспортни корисник транспортној целини да прихвати захтев за успоставом везе.

Раскид везе се извршава на сличан начин. Било која страна, или обе, могу да иницирају окончање, тј. затварање (Close). Веза се затвара обостраном сагласношћу.

Постоје два начина да се прекине веза а то су: изненадан прекид, када једна страна покрене иницијативу за прекид везе не сачекавши да то друга страна одобри, или прекид везе који настаје као резултат договора обе стране**.** Ако се веза изненада прекине, подаци који се у том тренутку налазе на путу између две станице могу бити изгубљени. Прекид везе уз међусобни договор обавезује обе стране да не прекида везу док се сви подаци не испоруче. Да би се то постигло, веза у стању FIN WAIT треба да настави да прима податке све док не добије *Fin* сегмент.

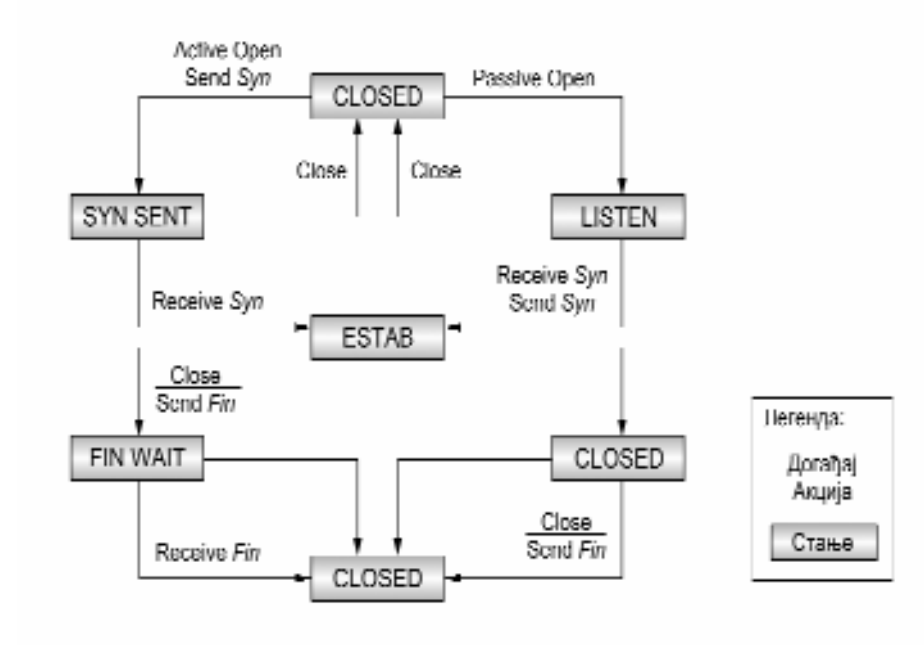
Слика 2.9 илуструје процедуру правилног прекида везе. Прво је неопходно уочити страну која иницира прекид везе:

- Као одоговор на примитиву Close корисника, транспортни целина шаље *Fin* сегмент другој страни, захтевајући прекид везе.
- Када пошаље *Fin* сегмент, транспортни целина ставља везу у FIN WAIT стање. Када је веза у овом стању, транспортна целина мора да настави да прихвата податке који стижу са друге стране и да испоручи податке кориснику.
- Када *Fin* сегмент стигне као одговор, транспортна целина затвара везу и о томе обавештава корисника.

Са тачке гледишта стране која није иницирала прекид, ситуација је седећа:

- Када страна која није иницирала прекид прими *Fin* сегмент, транпортна целина информише корисника да је иницран прекид везе и ставља везу у стање CLOSE WAIT. Када се веза налази у овом стању, транспортна целина мора да настави да прима податке од свог корисника и да их прослеђује ка другој страни.
- Када корисник изврши Close примитиву, транспортна целина шаље одговарајући *Fin* сегмент другој страни и затвара везу.

Ова процедура обезбеђује да се обе стране сложе око затварња везе и да приме податке који се налазе на линији, пре самог затварања везе, да не би дошло до губљења података.



Слика 2.9 Поједностављен дијаграм стања везе

Транспортни протокол и непоуздана мрежна услуга

Сложенији случај транспортног протокола је када је услуга мрежног слоја непоуздана. Примери мрежа које користе овакве услуге су:

- мреже које користе Интернет протокол (IP),
- мреже са штафетним преносом које користе само LAPF протокол,
- локалне рачунарске мреже Етернет типа[1] које на слоју везе користе услугу без успоставе везе и без потврде.

Проблеми су:

- повремени губитак сегмената,
- њихово пристизање ван послатог редоследа као последица променљивог кашњења кроз мрежу.

Потребно је анализирати:

- стратегију поновног слања,
- детекцију дуплих пакета,

[1] Стандард IEEE802.3

- контролу тока,
- успоставу везе,
- окончање везе,
- опоравак после испада из рада.

Стратегија поновног слања (ретрансмисија)

Два догађаја указују да је потребна стратегија за поновно слање сегмената. Прво, сегмент може да буде оштећен при проласку кроз мрежу али и поред тога стиже на одредиште. Уколико сегмент садржи контролу грешке пријемна транспортна целина може да детектује грешку и одбаци сегмент.

Друго, сегмент може да не стигне на одредиште. Транспортна целина која је послала сегмент не зна да је слање неуспешно. Да би се овакви проблеми разрешили уводи се потврда (*Ack*). Пријемник мора да потврди сваки успешно примљени сегмент. Због ефикасности не користи се један *Ack* по једном сегменту већ кумулативна потврда – један *Ack* за више сегмената. Значи, пријемник прима сегменте број 1, 2 и 3 али враћа натраг само *Ack 3*.

Уколико сегмент није успешно примљен, неће бити послата потврда (*Ack*) и долази до ретрансмисије. Ретрансмисија може бити селективна (поново се шаљу само неисправни пакети) или сви од неисправног па до оних који су послати до пријема негативне потврде[1].

За свако слање сегмената мора да постоји часовник[2] који се поставља на одређено време[3]. Ако то време истекне а сегмент није потврђен пошиљалац мора поново да шаље пакет.

Увођење часовника донекле разрешава проблем. Поставља се питање на коју вредност треба поставити часовник? Уколико је вредност превише мала постојаће много беспотребних ретрансмисија. Уколико је вредност превелика протокол ће бити инертан у реаговању на изгубљене сегменте. Часовник треба да буде постављен на вредност мало већу од укупног кашњења кроз мрежу[4] које је једнако збиру времена потребног да сегмент стигне на одредиште и да се врати његова потврда. Кашњење је променљиво и код мрежа са константним саобраћајем а нарочито код мрежа са променљивих карактеристика.

Постоје два метода: са тачном (фиксном) вредношћу и са променљивом (адаптивном, самоподешавајућом) вредношћу. Тачно одређена вредност на коју се часовник подешава

---

[1] Негативне потврде (*NACK- Negative ACKnowledge*) или поновљена потврда убрзавају поступак ретрансмисије – не чека се истицање времена на који се поставља часовник.
[2] *Timer*
[3] *Time out*
[4] *Round trip delay*

заснована је на уобичајеном стању мреже. Проблем са овим методом је што не одговара мрежама чије се стање мења. Уколико је вредност превише велика услуга ће увек бити инертна. Уколико је вредност превише ниска јавља се позитивна повратна спрега – загушење мреже које доводи до више ретрансмисија које повећавају саобраћај у мрежи и још веће загушење.

Адаптивна шема има својих ограничења. Претпоставимо да транспортни слој прати колико времена је потребно да би се потврдио сегмент и подешава часовник према праћеној усредњеној вредности. Може се десити да:

- одредиште не шаље потврду одмах (нпр. користи се кумулативна потврда),
- сегмент се шаље поново али пошиљалац не зна да ли је примљена потврда (*Ack*) за првобитно слање или за ретрансмисију,
- услови у мрежи могу се нагло променити.

Сваки од ових проблема се може разрешити на неки начин али свако од решења оставља одређени степен неодређености. Часовник за ретрансмисију је веома важан за исправан рад транспортног протокола. У табели 2.4 описани су сви часовници које транспортни протокол користи.

Часовник	Намена
За ретрансмисију[1]	Поново шаље сегмент који није потврђен.
За поновну успоставу везе[2]	Минимално време између затварања једне везе и отварања друге везе са истом одредишном адресом.
За отвор клизајућег прозора[3]	Максимално време узмеђу два *Ack* сегмента.
За ретрансмисију *Syn* сегмента[4]	Време између два покушаја да се веза успостави (отвори).
За прекид[5]	Прекида везу пошто ниједан сегмент није потврђен.

Табела 2.4 Часовници који се користе на транспорном слоју

Откривање (детекција) дупликата

Уколико се деси да је сегмент изгубљен и поново послат не може да дође до грешке. Уколико је *Ack* изгубљен један или више сегмената ће бити поново послати и уколико стигну успешно биће дупликати претходно послатих сегмената. Потребно је пронаћи начин да пријемник препозна дупликате. Постоје два случаја:

- дупликат је примљен пре затварања везе,

[1] *Retransmission timer*
[2] *Reconnection timer*
[3] *Window timer*
[4] *Retransmit Syn timer*
[5] *Give up timer*

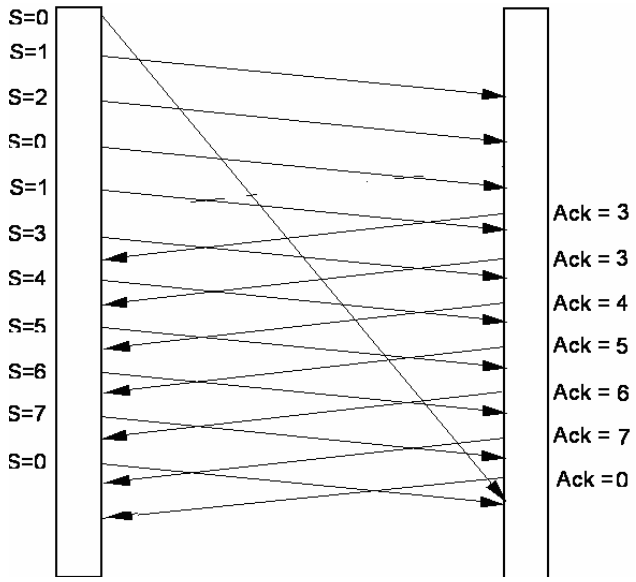
- дупликат је примљен по затварању везе.

Може да се деси да дупликат стигне пре оригиналног сегмента. Уколико се деси да дупликат стигне пре затварања везе онда:

- пријемник треба да претпостави да је потврда изгубљена и да је потребно да пошаље потврду за дупликат,
- простор редних бројева (број редних бројева[1]) треба подесити да је довољно велики.

Слика 2.10 приказује разлог за захтеве који следе. У овом примеру простор редних бројева је 8. Због једноставности претпостављамо да је отвор клизајућег прозора 3. Нека је станица 1 послала сегменте са редним бројевима 0, 1, 2 и није добила потврду о њиховом успешном пријему. Оглашава се часовник за ретрансмисију (пошто је време на које је подешен истекло). Станица 1 поново шаље сегмент 0. Станица 2 прима сегменте 1, 2 али је при проласку кроз мрежу сегмент 0 је закашњен. Када дупликат сегмента 0 стигне станица 2 потврђује сегменте 0, 1 и 2. У међувремену истекло је и време за пакет са редним бројем 1 па га станица 1 поново шаље. Станица 2 га потврђује новим *Ack2*. Када је простор редних бројева потпуно искоришћен 1 се поново враћа на редни број 0 и наставља даље да додељује редне бројеве. Али у међувремену стиже стари сегмент 0 пре него што је стигао нови сегмент 0.

[1] Поруку коју транспортна целина добија од свог корисника (апликационог слоја или слоја сесије) и коју треба да пошаље она може да подели у мање делове и шаље као посебне сегменте. Сваки од тих делова добија свој број. Ти бројеви се у литератури означавају као редни (секвенцни) бројеви. Простор редних бројева (укупан број) зависи од броја битова који се користи. На пример ако је додељено поље од 3 бита онда је простор редних бројева $2^3 = 8$. То зачи да сегменти могу да буду означени редним бројевима 0,1,...7.



Слика 2.10 Пример погрешне детекције дупликата

Јасно је да пристизање старог сегмента 0 „у невреме" не би изазвало никакве проблеме да није кренуо нови круг редних бројева. Питање је колико треба да буде велики простор редних бројева? Додавањем само једног бита у поље редних бројева може се удвостручити простор редних бројева. Стандардни транспортни протокол омогућава огроман простор редних бројева.

Анализираћемо други проблем: сегмент кружи мрежом по затварању транспортне везе. Ако је следећа веза отворена између истих транспортних целина сегмент из старе везе може да се појави и да буде прихваћен. Такође закаснела потврда може да се појави у новој транспортној вези и изазове проблеме.

Постоји неколико метода који могу да разреше ове проблеме. Анализираћемо неке од њих. Први метод се састоји у следећем: транспортна целина памти последњи реднини број који је употребљен у завршеној транспортној вези. Када се нова веза успоставља *Syn* садржи редни број са којим треба да се започне пренос података.

Други приступ је увођење посебног идентификатора транспортне везе и употреба новог идентификатора са сваком новом везом.

Описана процедура ради добро докле год не дође до пада система, па се информација о идентификатору последње везе губи. Алтернатива је да се једноставно сачека довољно дуго између две транспортне везе да би осигурали да су сви сегменти испоручени. Уколико дође до пада система на једној страни друга страна може да одбије успостављање везе док време на часовнику за поновну успоставу везе не истекне. Ово наравно може да изазове штетна кашњења.

Контрола тока

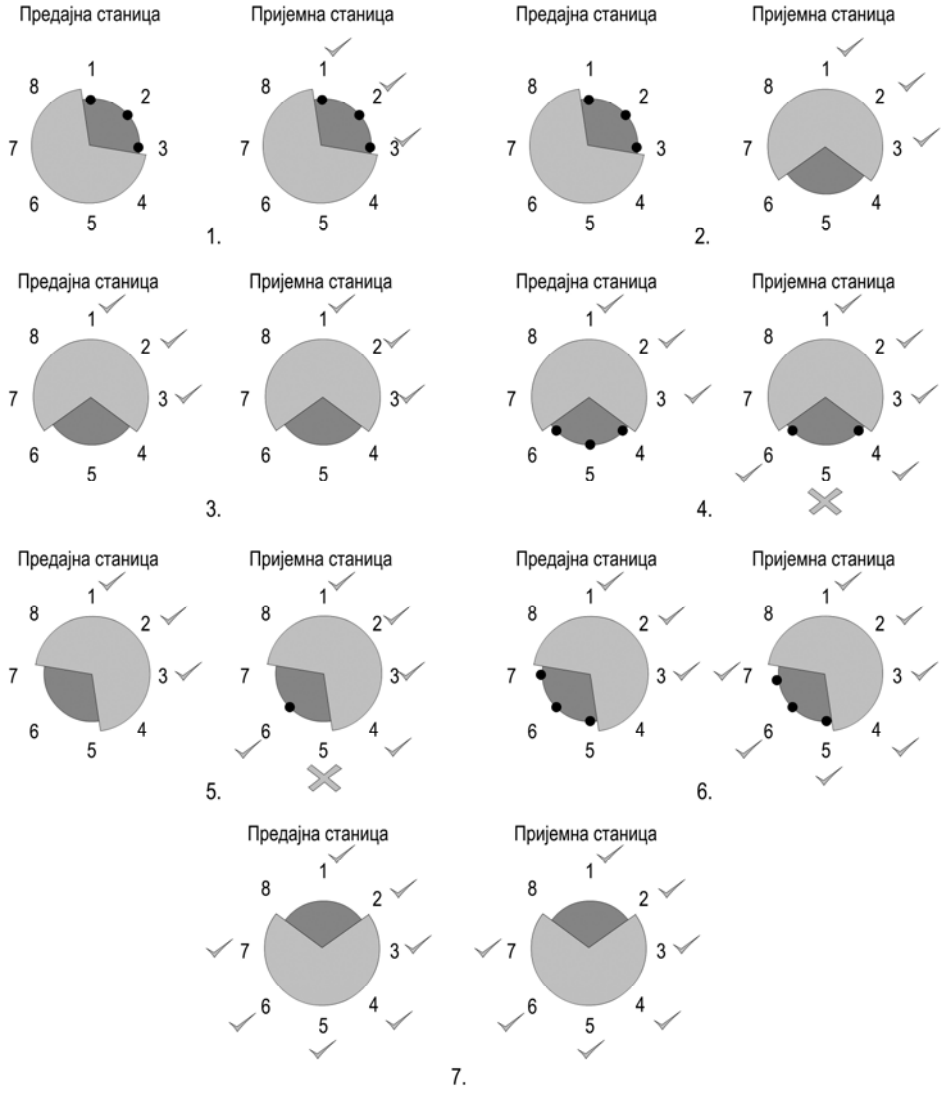
Контрола тока са механизмом кредита представља добро решење за мреже са непоузданом услугом мрежног слоја. Механизам са доделом кредита и механизам потврде повезани су тако да пријемна транспортна целина шаље управљачки сегмент који садржи:

- потврду о успешном пријему сегмента закључно са редним бројем *N-1* (означава се са *Ack N*),
- дозволу да предајник пошаље сегмент почевши од редног броја *N* до редног броја *N+ M* (означава се са *Credit M*).

Уколико се деси да потврда не стигне на време, часовник за ретрансмисију ће то сигнализирати и предајник ће поново послати сегменте који захтевају нову потврду. Грешка је и даље могућа али се може превазићи, што ћемо видети у следећем примеру:

- станица 2 (пријемна станица) шаље сегмент са параметрима (*Ack N, Credit M*) и привремено затвара прозор,
- затим шаље сегмент са параметрима (*Ack N, Credit 0*) али овај сегмент је изгубљен,
- станица 1 чека дозволу да пошаље податке, станица 2 зна да је дозвола послата,
- часовник за отвор прозора (активира се при слању сваког сегмента *Ack*/*Credit*) сигнализира да је време истекло,
- понавља се последњи *Ack*/*Credit* сегмент и указује пошиљаоцу да је одредиште и даље активно.

На слици 2.11 је приказан пример контоле тока са механизмом кредита. Види се да се доња граница отвора прозора поставља када се сегмент пошаље а горња граница када се добије сегмент са кредитом.



Слика 2.11 Пример технике клизајућег прозора са кредитом

Објашњење примера са слике 2.11

1. Отвор прозора је подешен на 3 и код предајне и код пријемне станице. Предајна станица шаље три сегмента пријемној станици.
2. Пријемна станица је успешно примила сва три сегмента и подешава свој отвор прозора за следећа три сегмента, а затим шаље предајној станици сегмент

потврде[1] означен као *Ack4* и кредит који је 3. Број 4 указује да је редни број првог следећег сегемнта који очекује 4.

3. На основу примљеног *Ack4* сегемнта, предајна станица зна да су прва три сегмента испоручена а на основу кредита (3) подешава отвор прозора за следећа три сегемнта: сегемнте са редним бројевима 4., 5 и 6 које затим шаље.
4. Пријемна станица прима сва три сегмента, али 5. сегмент има грешку. Пријемна станица помера свој отвор прозора за један и шаље предајној станици сегмент *Ack5* и кредит који је 3. На тај начин обавештава предајну станицу да је примила 4. сегмент, и да поново пошаље 5., 6., и 7.
5. Предајна станица прима *Ack5* сегмент и прилагођава свој отвор прозора. Затим шаље сегмент са редним бројевима 5, 6 и 7.
6. Пријемна станица успешно прима 5., 6. и 7. сегмент, подешава свој отвор прозора на нову вредност, а затим предајној станици шаље потврду да је успешно примила сегменте и нови кредит.
7. Предајна станица добија потврду о пријему сегмената и подешава свој отвор прозора на нову вредност.

Успостављање везе

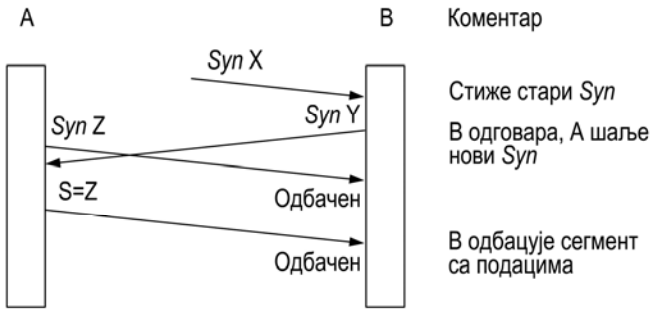
Успостављање везе као и остали механизми протокола треба да узму у обзир непоуздану услугу мрежног слоја. Размена *Syn* [2] сегмената некада се назива двострана процедура[3]. Претпоставимо да станица 1 шаље *Syn* сегмент ка станици 2. Очекује да добије натраг потврду да је веза успостављена. До грешке може да дође ако се деси следеће: сегмент *Syn* који шаље станица 1 је изгубљен, или је сегмент *Syn* који као одговор шаље станица 2 изгубљен. Оба случаја могу се превазићи са часовницима за поновно слање (ретрансмисију).

Поновна слања могу довести до дуплирања *Syn* сегмената. Када је веза успостављена, станице 1 и 2 могу једноставно да игноришу дупликате.

Шта се дешава уколико дупликати *Syn* сегмента опстану по завршетку везе? На слици 2.12 приказан је проблем који може да се појави. Стари *Syn* сегмент (захтев за успоставом везе, редни број почиње са *X*) стиже до станице 2 пошто је веза окончана. Станица 2 претпоставља да је ово нови захтев и одговара са *SynY*. У међувремену станица 1 је одлучила да успостави нову везу са станицом 2 и шаље *Syn Z*. Станица 2 га

[1] Потврда о успешном пријему означава се са *Ack* (*Positive Acknowledge* - позитивна потврда). Уколико је примљен пакет са грешком онда одредиште шаље пакет који указује да је примљени пакет са грешком *NAck* (*Negativ Acknowledge* - негативна потврда).
[2] Код TCP протокола означава се са *Syn.* Користи се и ознака CR TPDU (*Connection Request Transport Protocol Data Unit*)
[3] *Two-way handshake*

одбацује као дупликат. Станица 1 започиње пренос података TPDU пакетом са редним бројем Z. Станица 2 га одбацује као пакет ван очекиваног опсега редних бројева.



Слика 2.12 Пример двостране процедуре

Начин за превазилажење овог проблема је да свака од страна потврди пријем *Syn* и редног броја друге стране. Процедура се назива тространa[1] а слика 2.13 илуструје типични начин рада са тространом процедуром. У нормалним околностима станица 1 шаље *Syn X* који садржи предајни редни број. Ова вредност представља иницијални редни број станице 1. Станица 2 у *Syn* сегменту који шаље као одговор садржи потврду примљеног редногног броја и свој иницијални редни број. Станица 1 првим сегментом података потврђује пријем *Synx*/*Ack* сегмента станице 2. На слици 2.13б приказана је ситуација у којој станици 2 стиже стари *Syn X* после окончања везе. Станица 2 претпоставља да је ово захтев за успоставом нове везе и одговара са *Syn Y*, *Ack X*. Када станица 2 добије овај сегмент, и види да није захтевала ову везу, шаље сегмент *Rst*, *Ack Y*. Уочимо да је *Ack Y* део у *Rst* сегемнту веома значајан, тако да стари *Rst* не може да прекида регуларну успоставу везе. Пример на слици 2.13в указује на случај када стари *Syn*/*Ack* стиже у току успоставе нове везе. Због употребе редних бројева у потврди овај догађај неће изазвати никакав проблем.

[1] *Three - way handshake*

Слика 2.13 Пример тростране процедуре у нормалном начину рада, са кашњењем *Syn* и *Syn*/*Ack* сегмената

Окончавање везе

Као и у случају успоставе везе процедура за окончавање везе мора да се бави старим и изгубљеним управљачким сегментима. Да би се спречила неусаглашеност, користе се редни бројеви на следећи начин:

- *Fin* сегмент садржи редни број последњег сегмента увећан за један,
- *Ack* сегмент садржи редни број примљен у *Fin* сегменту.

Процедуре које се покрећу када истекне време на часовницима користе се за затварање везе када друга страна везе не ради исправно.

Опоравак после отказа[1]

Када систем на коме ради посматрана транспортна целина престане са радом (падне) и после се поново покрене, информације о стању свих веза су изгубљене. Везе се сматрају полуотвореним пошто друга страна (код које није дошло до грешке) није још сагледала проблем.

Још увек активна страна полуотворене везе може да је затвори користећи часовник за прекид[2]. Часовник мери време које истекне после максималног броја ретрансмисија. По истицању тог времена транспортна целина претпоставља да је друга транспортна целина у квару и обавештава транспортног корисника о нерегуларном окончању везе.

У случају да је транспортна целина кратко време у квару па убрзо поново почне да ради, полуотворена веза се може окончати употребом *Rst* сегмента. Страна која је у квару шаље *Rst X* за сваки сегмент података редног броја x који прими. Када *Rst X* стигне на другу страну мора се проверити на основу редног броја *X* пошто *Rst* може да буде одговор на стари сегмент. Уколико је прекид важећи транспортна целина ће извршити нерегуларно окончање везе.

[1] *Crash recovery*
[2] *Give up timer*

3. Транспортни слој на Интернету

Протокол TCP је веома сложен протокол пројектован тако да обезбеди поуздан с краја на крај проток бајтова. Сваки рачунар који подржава TCP протокол поседује TCP транспортну целину као библотечку процедуру, кориснички процес, или део језгра оперативног система. У сваком случају обезбеђује проток TCP сегмената и интерфејс до мрежног слоја. Прво ћемо анализирати рачунарске мреже у којима услуга мрежног слоја гарантује испоруку свих транспортних јединица података у послатом редоследу. Затим ћемо анализирати механизам транспортног протокола који се захтева у рачунарским мрежама са непоузданом мрежном услугом као што је Интернет.

Поуздана услуга мрежног слоја

Претпоставићемо да мрежни слој прихвата поруке различите дужине и да их са великом поузданошћу испоручује одредишту у правилном редоследу. Примери комуникационих мрежа које обезбеђују овакву услугу су:

- X.25 мрежа са комутацијом пакета високе поузданости,
- мрежа са штафетним преносом која користи LAPF[1] протокол,
- локалне рачунарске мреже по стандарду IEEE802.3 и услугом са успоставом везе коју обезбеђује подслој за управљање везом (LLC[2]).

У наведеним примерима TCP протокол се користи као протокол с краја на крај, између два система која су везана за исту комуникациону мрежу, а не као протокол за везу између два система везана за међусобно повезане мреже тј. Интернет.

Када се користи комуникациона мрежа са поузданом услугом мрежног слоја на транспортном слоју се може користити једноставан протокол. Четири функције које би поуздан протокол требало да подржи су:

- адресирање,
- мултиплексирање,
- контрола тока,
- успостављање и раскидање везе.

Адресирање

Да би одредишна станица била у потпуности одређена потребне су следеће информације:

---

[1] *Link Access Procedure for Frame Mode Bearer Services*
[2] *Logical Link Control*

- идентификација корисника,
- идентификација транспортне целине,
- адреса станице,
- број мреже.

Поставља се питања како изворишна станица зна адресу одредишне станице? Постоје четири начина да се овај проблем реши:

- ТС корисник зна адресу одредишта - постиже се конфигурацијом система,
- често коришћене услуге су означени као „добро познате адресе"[1]. На пример FTP, SMTP услуге на страни сервера и неки други стандардни протоколи. У табели 3.1 приказани су бројеви портова који се најчешће користе,
- сервер имена[2],
- изазвани процес[3] (слика 2.5).

Порт	Протокол	Намена
21	FTP	Пренос датотека
23	Telnet	Виртуелни терминал
25	SMTP	Електронска пошта
69	TFTP	Trivial File Transfer Protocol
79	Finger	Преглед информација о корисницима
80	HTTP	Word Wide Web
110	POP-3	Удаљени приступ електронске поште
53	DNS	Сервер имена домена

Табела 3.1 Најчешће коришћени портови

Мултиплексирање

Транспортни протокол обавља функцију милтиплексирања/демултиплексирања (слика 2.6):

- када више транспортних корисника дели једну виртуелну везу успостављену између мрежних целина,
- када једна транспортна веза користи више веза на мрежном слоју.

Контрола тока

[1] *Well-known address*. Листа се може наћи на *www.iana.org*.
[2] *Name server*
[3] *Spawned*

Контрола тока је на слоју везе једноставан механизам док је на транспортном слоју знатно сложенији[1]. Два разлога доводе до тога да једна TCP целина ограничава брзину преноса сегмената ка другој TCP целини:

- корисник пријемне TCP целине не може да подржи брзину пристизања података.
- пријемна TCP целина не може да подржи брзину пристизања сегмената.

Постоји више начина које може да примени транспортна целина. У овом делу анализираћемо једну од њих - механизам кредитирања који се користи код TCP протокола. Механизам кредитирања користи:

- редне бројеве,
- отвор (ширина) прозора за слање,
- потврду о пријему пакета која доприноси повећању поузданости и служи за подешавање величине прозора.

Код механизма кредитирања сваки индивидуални бајт података има јединствен редни број и као такав се преноси. Такође, сваки сегмент који се шаље у свом заглављу има три поља која се односе на контролу тока:

- редни број означен са *X*,
- број потврде означен са *Y*,
- отвор прозора или кредит означен са *Z*.

Када транспортна целина шаље сегмент, она у заглавље уписује редни број првог бајта (октета). Пријемна транспортна целина потврђује пријем тако што шаље сегмент код кога су постављене вредности $Y = i$, $Z = j$ што значи:

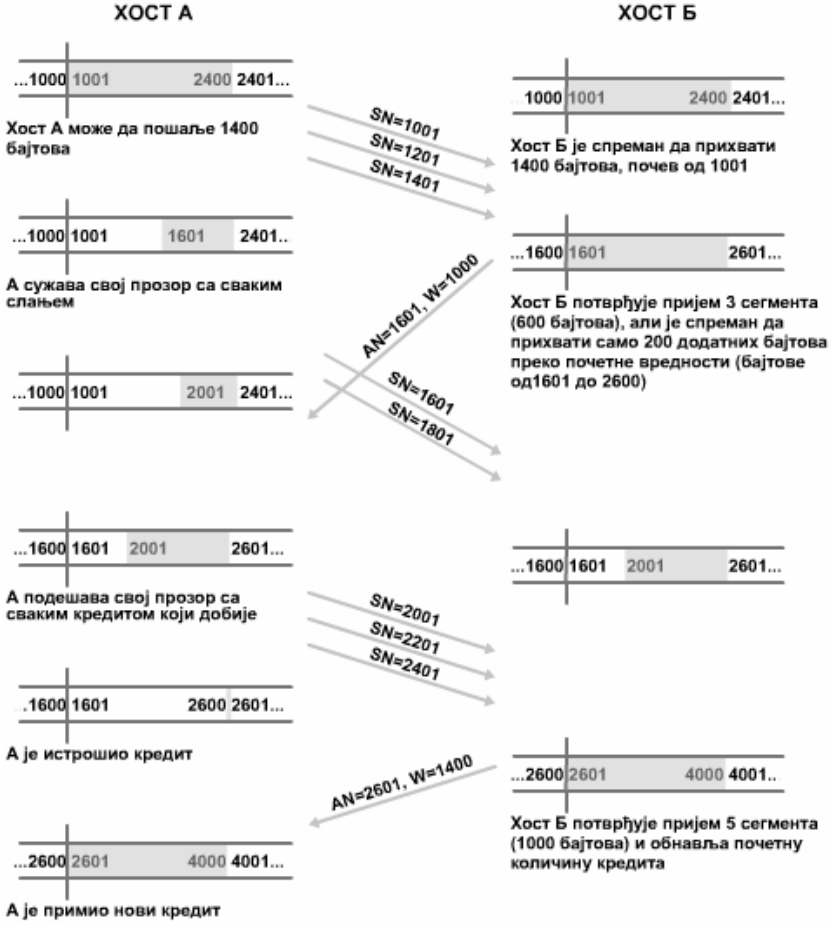
- сви бајтови закључно са бројем $Y = i - 1$ су примљени, и очекује се сегмент чији је први бајт означен са редним бројем $Y = i$.
- дозвољава се предајнику да је његов отвор прозора величине $Z = j$ бајтова.

Очекује се да ће редни бројеви бити у опсегу од $X = i$ до $X = i + j - 1$.

Слика 3. 1 илуструје овај механизам. Због једноставности приказан је проток података у једном смеру и претпостављена је величина сегмента од 200 бајтова. У току трајања

[1] Објашњено у поглављу о транспортном слоју.

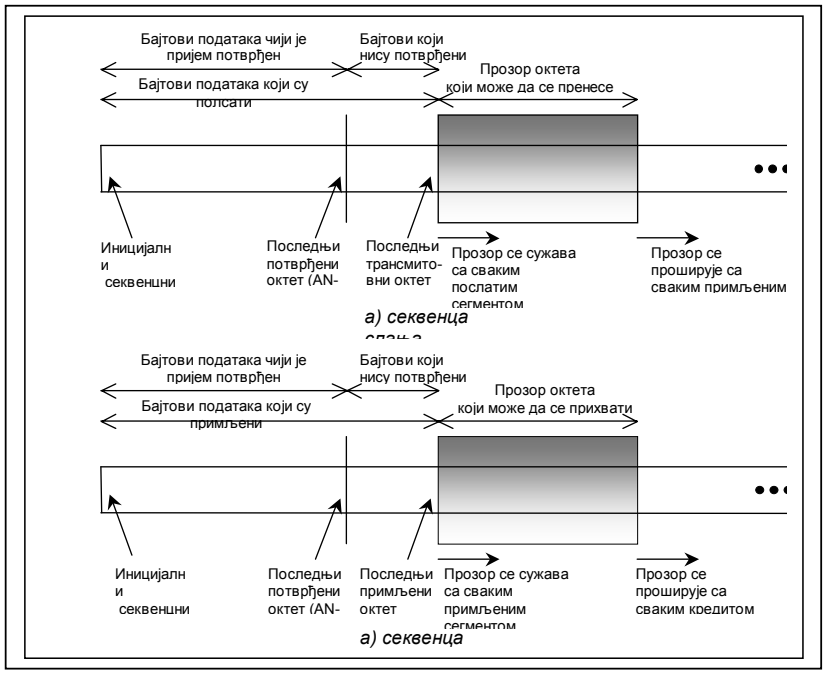
процеса успоставе везе пријемни и предајни редни бројеви су синхронизовани.



Слика 3.1. Пример механизма за доделу кредита код TCP протокола

Рачунар (станица 1) добио је кредит од 1400 бајтова, почев од бајта са редним бројем 1001. После слања 600 бајтова, у по три сегмента, станица 1 сужава свој прозор на 800 бајтова (редни бројеви од 1601 до 2400). Када станица 2 прими ова три сегмента њој остаје кредит од 800 бајтова. Претпоставимо да станица 2 сада може да прихвати 1000 бајтова са ове везе. Према томе, станица 2 је потврдила пријем свих бајтова до редног броја 1600 и послала кредит од 1000 бајтова. То значи да станица 1 може да пошаље бајтове од 1601 до 2600 (5 сегмената, 1 сегмент = 200 бајтова). За време за које је сегмент станице 2 са додатним кредитом стигао до станице 1, станица 1 је већ послала два сегмента са редним бројевима од 1601 до 2000 (почетна величина прозора). Према томе, станици 1 може да пошаље 600 бајтова (3 сегмента, са редним бројевима од 2000 до 2600).

У току размена података станица 1 смањује величину свог прозора сваки пут када пошаље сегмент, а повећава отвор свог прозора када прими сегмент са додатним кредитом.



Слика 3.2 Контрола тока при слању и пријему сегмената

Пријемник је у обавези да усвоји неки скуп правила која ће одређивати колико података пошиљалац може да пошаље примаоцу. Један приступ био би да пошиљалац прима нове сегменте само до тренутка док они не попуне меморијски простор (бафер). Ако овај приступ применимо на пример приказан на слици 3.1 онда би први сегмент са кредитом коју пошаље станица 2 указивао да она има на располагању слободан меморијски простор за 1000 бајтова, а други за 1400 бајтова. Слика 3.2 приказује како се механизам клизајућег прозора реализује на предајној и пријемној страни.

У случају да је пошиљалац бржи од примаоца неке сегменте прималац ће одбацити, што ће довести до потребе за поновним слањем (ретрансмисијом). У комуникационој мрежи није пожељно поновно слање сегмената.

Успостављање и раскидање везе

Успостављање везе има за циљ да се крајеви TCP везе увере да постоје, да се договоре око параметара као што је нпр. величина прозора и да се доделе ресурси TCP целини (нпр. меморијски простор).

Раскидање везе обавља се сагласношћу обе стране. У зависности од начина како се обави може да дође:

- до губитка података чији је пренос у току код наглог раскида везе или,
- до одлагање раскида везе док се сви подаци чији је пренос у току не испоруче одредишту.

Описани приступ за контролу тока може ограничити пропусност TCP везе када постоје велика кашњења. Прималац може да повећа пропусност тако што ће пошиљаоцу понудити већи кредит него што је величина слободног меморијског простора. На пример прималац ће послати пакет са кредитом величине 1000 бајтова иако је његов меморисјки просто попуњен. Прималац претпоставља да може да ослободи меморијски простор од 1000 бајтова за време које протекне од тренутка када пошаље сегемнт са кредитом и до тренутка када од пошиљаоца добије нове податке.

Услуга транспортног слоја

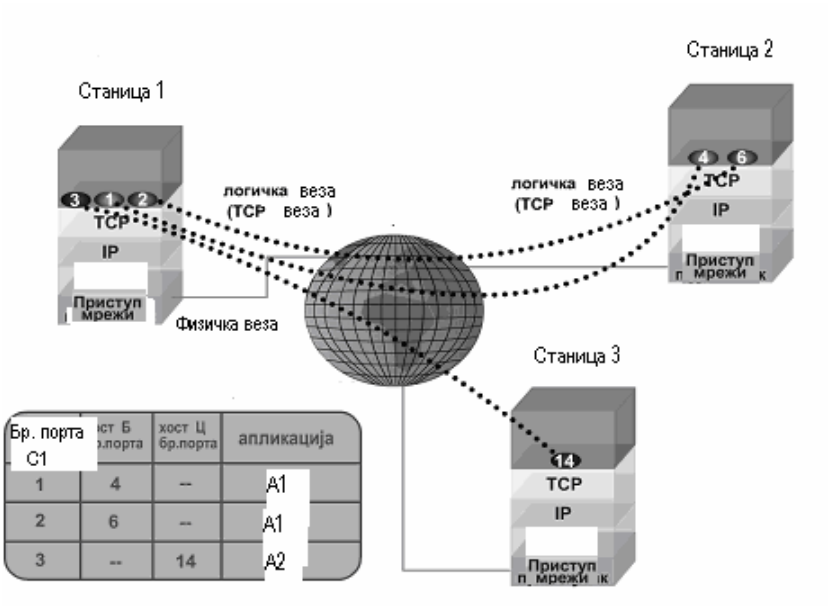
Да би се обезбедила услуга транспортног слоја (TCP услуга) морају се дефинисати две прикључне тачке[1]. Једна је креирана од стране пошиљаоца, а друга од стране примаоца. Свака прикључна тачка има свој број (адресу) који се састоји од пара (станица, порт). Порт је назив који се користи за приступну тачку услузи транспортног слоја (TSAP[2]). У случају једне рачунарске мреже (нпр. локалне рачунарске мреже) станицу одређује мрежни уређај којим се прикључује на локалну рачунарску мрежу. Када је рачунарска мрежа Интернет станица представља глобалну адресу на Интернету. Да би се реализовала TCP услуга, мора се успоставити веза између прикључне тачке на рачунару пошиљаоца и прикључне тачке на рачунару примаоца.

Крајње тачке могу се користити за више веза истовремено. Другим речима, две или више веза могу се завршавати на истој крајњој тачци. Везе су одређене идентификаторима (прикључницама) на оба краја који су означени као (*socket1*, *socket2*).

На слици 3.3 приказане су три станице повезане на Интернет. Свака од станица има јединствену Интернет адресу. TCP целина у оквиру станице опслужује више портова а свакој од апликација додељен један или више портова. У овом примеру постоје две активне TCP везе: између апликације 1 (A_1) на станици 1 (C_1) и апликације 2 (A_2) на станици 2 (C_2). Такође постоји веза између апликације A_1 на станици 1 и апликације A_2 на станици 3 (C_3). TCP целина може да:

- истовремено подржава више веза са апликационим слојем и
- истовремено подржава више веза са истом апликацијом на апликационом слоју.

[1] *Socket*
[2] *Transport Service Access Point*



Слика 3.3. Пример мултиплексирања код TCP заснованог на броју порта

Портови чији су бројеви мањи од 1024 називају се познати портови и резервисани су за стандардне услуге (табела 3.1). На пример: било који процес који жели да успостави везу са неком станицом и преноси датотеке користећи FTP протокол може да се повеже са портом 21 удаљене станице да би успоставио везу са његовим FTP демоном[1]. Све TCP везе су двосмерне (потпуни дуплекс) и тачка-тачка. Двосмерна веза указује да постоји двосмерни проток података а тачка-тачка да свака веза има само две крајње тачке. TCP протокол не подржава везе „један ка групи"[2] нити „један ка свима"[3].

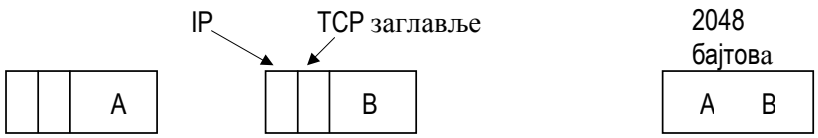
За TCP услуге ситуација је сложенија када се она ослања на непоуздану мрежну услугу. Пример за то су:

- мреже са IP протоколом,
- локалне рачунарске мреже са IEEE802.3 протоколом и услугом без успоставе везе LLC подслоја.

TCP веза представља проток бајтова, не проток порука. Границе порука нису очуване при преносу са једног на други крај. На пример, ако TCP пошиљалац шаље четири податка од по 512 бајтова они могу бити уручени апликацији примаоца као четири дела од 512 бајтова, два дела од 1024 бајта или као један део од 2048 бајтова. На слици 3.4 приказана је испорука два дела од 1024 бајта као један део од 2048 бајтова. Не постоји

[1] Процес који се у позадини, невидљиво, извршава.
[2] Мултикастинг
[3] Бродкастинг

начин да прималац сазна у којим јединицама је порука послата. TCP софтвер не зна шта ти бајтови значе и нема интерес да то сазна.



Слика 3.4 Испорука података апликацији: два послата дела од 1024 бајта испоручују се као један део од 2048 бајтова

Када апликација проследи податак TCP процесу, он га може послати или сместити у прихватну меморију у намери да сакупи већу количину података и пошаље их одједном. Понекад апликација заиста жели да подаци буду одмах послати. На пример, претпоставимо да је корисник пријављен на удаљеном рачунару. Након што се изврши једна командна линија и откуца прелазак у нови ред[1] битно је да се командна линија пошаље што пре удаљеном рачунару и проследи пре него што нова командна линија пристигне. Да би апликација убрзала прослеђивање података она ће указати одговарајућим механизмом TCP целини да не треба да задржава те податке.

Последња битна примена TCP услуге, коју ћемо поменути, је употреба података које хитно треба проследити[2]. Када корисник притисне тастере DEL или CTRL-C да би прекинуо рад на удаљеном рачунару, апликација за слање додаје контролне информације у низ података и обавештава на одговарајући начин TCP процес на пријемној страни да заустави нагомилавање података и одмах корисницима (апликационим процесима) проследи све податке које има за ту везу.

3.2 Преглед TCP протокола

У овом одељку даћемо општи преглед TCP протокола. Слике 3.5 и 3.6 илуструју основне операције које извршава TCP протокол. Корисник шаље податке TCP целини, део по део, помоћу примитива за слање[3]. Подаци се смештају у резервисани меморијски простор[4] за слање. С времена на време, TCP узима податке из меморијског простора и пакује у сегменте који се даље преносе. Сваки сегмент се прослеђује одредишној TCP целини (и даље кориснику) помоћу услуге мрежног слоја (IP услуге). Када одредишна TCP целина прими податке она уклања заглавље са сегмента а корисничке податке ставља у пријемни меморијски простор. Пријемна TCP целина помоћу примитиве за испоруку[5] повремено обавештава корисника да су подаци стигли и да су спремни за прослеђивање.

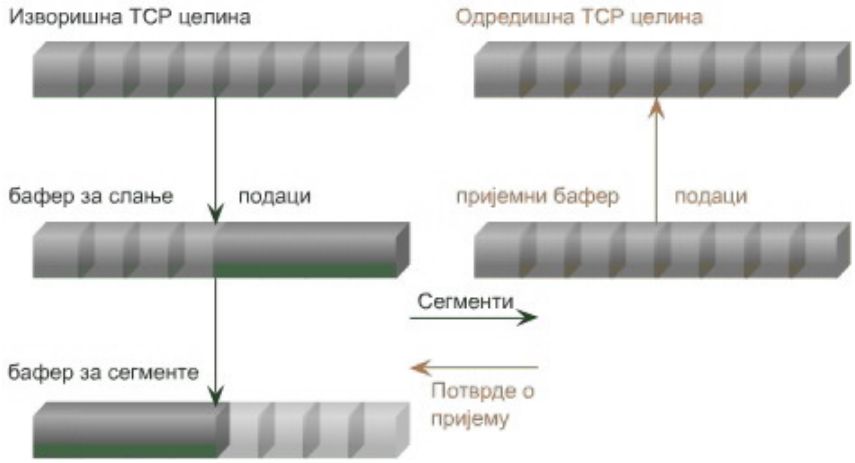
---

[1] *Carriage Return*
[2] *Urgent data*
[3] *Send Primitives*
[4] Бафер
[5] *Deliver*

Сваки бајт на TCP вези има свој сопствени 32-битни редни број. Он се користи (као што је већ објашњено) код механизма потврде и механизма клизајућег прозора. Предајна и пријемна TCP целина[1] размењују податке у облику сегмента. Сегмент се састоји од двадесетобајтног заглавља, опциони дела и (или не) бајтова података. TCP софтвер одлучује колике ће величине бити сегмент. Може сакупити податке у један сегмент или их поделити у неколико сегмената. Постоје ограничења која утичу на величину сегмента.

Један од пробелма који може настати при преносу података је да подаци могу стићи на одредиште у другачијем редоследу од онога у којем су послати. На основу редних бројева смештених у TCP заглављу сваког сегмента одредишни TCP целина лако може да уочи прави распоред сегемната, да их потом пресложи и испоручи кориснику

Други проблем који може да настане је ако се сегмент изгуби при преносу. Овај проблем се превазилизи употребом потврда о пријему сегемната, чиме се омогућава да TCP целине могу да поново пошаљу изгубљене сегменте. Да би TCP био могао да поново пошаље изгубљене сегменте, он мора да поседује копију сваког послатог сегмента у за то намењеном меморијском простору, све док не добије потврду да је сегмент успешно испоручен (3.5).



Слика 3.5 Основне TCP операције

Рад са отвором прозора код TCP није директно везан за механизам потврде као што је случај са протоколима слоја везе што се може уочити у примеру који следи. Претпоставимо да прималац има на располагању меморијски простор од 9192 бајтова, као што је приказано на слици 3.6. Ако се пошаље сегмент од 4096 бајтова и на одредишту исправно прими, прималац ће потврдити сегмент. Сада је на располагању меморијски простор од 4096 бајтова (све док апликација не уклони неке податке) па ће

[1] Ентитет

пријемник обавестити предајник да је отвор прозора величине 4096 бајтова почевши од следећег бајта који очекује.

Пошиљалац шаље других 4096 бајтова, који су потврђени, али објављени отвор прозора сада је нула. Пошиљалац мора да прекине слање док процес апликације на страни примаоца не очита неке податке из меморијског простора. После очитавања, TCP целина може послати сегмент (објавити) којим се отвора прозор.

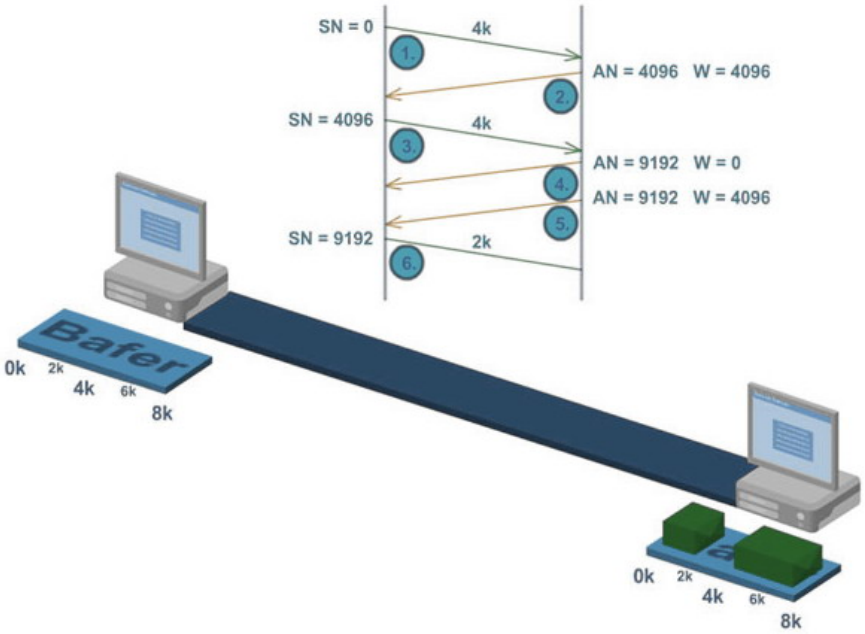
Када је отвор прозора једнак нули пошиљалац не може да шаље сегменте осим када је потребно послати:

- хитне податке или
- сегмент са податком од једног бајта са потврдом и променом величине отвора прозора за супротни смер везе.

TCP стандард дозвољава ову опцију да би спречио потпуни застој[1] који би настао у случају да се обавештење о отвору прозора изгуби.

Од пошиљаоца се не захтева да проследи података чим их добије од апликације. Нити се од примаоца очекује да одмах пошаље потврду. Као што је приказано на слици 3.6, након доласка првих 4kB[2] података TCP целина на пријему, знајући ди има на располагању величину отвора прозора од 8kB, може да складишти податке док не стигне још 4kB података.

[1] *Deadlock*
[2] 4kB = 4*1024 бајтова.



Слика 3.6 Меморијски простор пријемника и отвор прозора

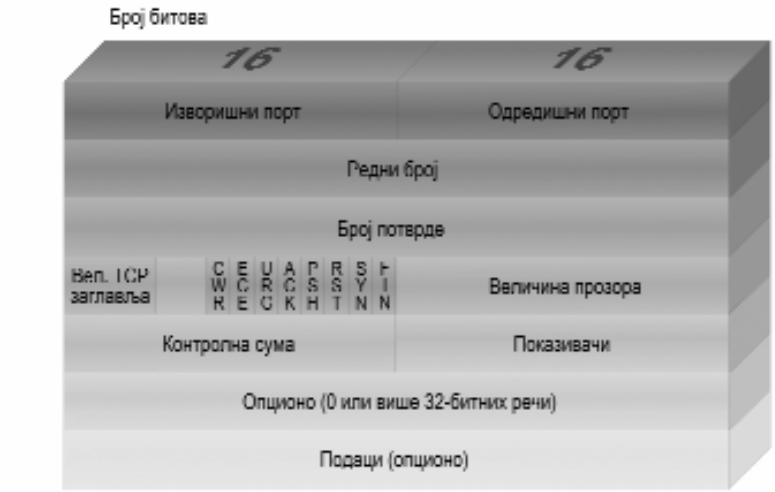
Comment [A1]: Page: 1
Продискутовати са Николом и Веселином овај цртеж

3.3 Формат TCP сегмента

Слика 3.7 показује формат TCP сегмента. Сваки сегмент почиње заглављем од 20 бајтова. Заглавље може бити праћено опционим делом. После опционог дела може се послати највише 65515 бајтова података[1]. Сегмент без података је могућ и обично се користи као потврда или управљање. У овом одељку анализираћемо садржај заглавља TCP сегмента.

Изворишни и одредишни порт[2] су поља дужине по 16 бита. Ова поља одређују крајње тачке везе. Сваки корисник може одлучити како да распореди сопствене портове почевши од 1024 па навише.

[1] 65535-40 = 65515; 20 (IP заглавље) + 20 (TCP заглавље без опционог дела) = 40 бајтова.
[2] *Source port, destination port*



Слика 3.7 Заглавље TCP сегмента

Редни број (SN[1]) је поље величине 32 бита. Представља редни број првог бајта података у том сегменту осим у случају да је постављен на јединицу SYN бит. Тада поље редни број означава почетни (иницијални) редни број ISN[2] тог правца везе. Првом бајту податка у том правцу везе биће додељен редни број ISN+1.

Број потврде (AN[3]) је поље величине 32 бита. Означава редни број податка који TCP целина очекује да прими.

Дужина заглавља[4] TCP сегмента је поље које носи информацију колико речи дужине 32 бита садржи заглавље. Ово поље је потребно да би пријемна страна знала величину заглавља пошто је поље опције променљиве величине.

Резервисано поље је величине 4 бита. Намењено је за будућу употребу и тренутно се не користи.

Ознака (флег) је поље величине 8 бита. Сваки бит у овом пољу има одређену функцију:

- CWR[1] (прозор загушења је смањен) постављен на 1 указује да је прозор загушења смањен.

[1] *Sequnce Number*
[2] *Initial Sequence Number*
[3] *Acknowledgement Number*
[4] TCP *header lenght*

- ECE[2] (exo експлицитног обавештавања о загушењу) заједно са ознаком CWR дефинисани су у документу RFC3168. Користе се за обавештавање о загушењу које ће бити објашњено у следећем поглављу.
- URG[3] постављен на 1 означава да је поље које указује[4] на место „хитних" података од значаја и у употреби. Функција која обезбеђује испоруку хитних података[5] служи да би се указало апликацији да су стигли хитни подаци и да одмах треба да јој се испоруче.
- ACK[6] постављен на 1 означава да је поље број потврде важеће и у употреби. Сегмент са постављеним ACK битом означава се као *Ack* сегмент. Ако је ACK=0 сегмент не садржи потврду. У том случају поље „потврдни број" се игнорише.
- PSH[7] поствљен на 1 од TCP целине - примаоца захтева да испоручује податке апликацији одмах по њиховом доласку и да их не складишти (што би он урадио због ефикасности).
- RST[8] постваљен на 1 користи се за ресетовање везе. Сегмент са постављеним RST битом означава се као *Rst* сегмент. Користи се и да би се одбацио неочекивани сегмент или одбио захтев за отварање нове везе.
- SYN[9] постављен на 1 користи се при успостављању везе. Сегмент са постављеним SYN битом означава се као *Syn* сегмент. Када се шаље захтев за успоставом везе, поље SYN и ACK имају вредности 1 и 0 респективно[10]. Када се шаље потврда на захтев за успоставом везе оба поља[11] имају вредности 1 а одговарајући сегмент означавамо као *Syn_Ack* сегмент.
- FIN[12] постављен на 1 користи се при ослобађању везе. Сегмент са постављеним FIN битом означава се као *Fin* сегмент. Указује да пошиљалац нема више података за слање.

Отвор прозора[13] (кредит) је поље величине 16 битова и користи се за контролу тока. Садржи број бајтова података, почев од редног броја који је наведен у пољу „потврдни број" које прималац може да прихвати. Када је вредност овог поља једнака 0 указује да прималац не може да прихвати више података.

---

[1] *Congestion Window Reduced*
[2] *Explicit Congestion Notification - Exho*
[3] *Urgent function*
[4] *Urgent Pointer*
[5] Са постављеним URG битом на вредност 1 и одговарајућом вредношћу у пољу *Urgent Pointer*.
[6] *Acknowledge*
[7] *Push*
[8] *Reset*
[9] *Synchronize*
[10] SYN=1, ACK=0
[11] SYN=1, ACK=1
[12] *Finish*
[13] *Window size*

Контролна сума[1] је поље величине 16 бита. Користи се да би се избегао пријем сегмената који су оштећени при преносу. Када пошиљалац шаље сегмент он израчуна контролну суму и упише је у ово поље. Када прималац прими сегмент он такође израчуна контролну суму и ако је вредност коју добије иста као и вредност поља контролне суме, значи да при преносу није дошло до грешке и сегмент се прихвата.

Контролна сума се израчунава као први комплемент суме првих комплемената шеснаестобитних речи псеудозаглавља, заглавља и података TCP сегмента. Псеудозаглавље укључује следећа поља из заглавља IP пакета: изворишну и одредишну IP адресу, поље које означава протокола[2] и поље које означава дужина TCP сегмента (слика 3.7). При израчунавању контролне суме вредност поља контролне суме поставља се на нулу. Разлог због кога TCP целина укључује и IP псеудозаглавље (само за време израчунавања) је што се на овај начин штити од грешака које могу да настану на мрежном слоју. У случају да се испоручи пакет погрешној станици, TCP целина ће препознати грешку.

Слика 3.8 Формат псеудозаглавља

Указивач „хитних" података[3] је поље величине 16 бита. Када се ова вредност дода на редни број TCP сегмента добија се вредност последњег бајта у секвенци података које треба хитно испоручити апликацији.

Опције[4] је поље променљиве величине. Користи се за додатне услуге који нису укључени у регуларно заглавље. Једна од опција је, на пример, величина сегмента која се користи при преносу. Требало би користити што веће сегменте[5] јер се на тај начин преноси више података[6] а заглавље од 20 бајтова представља његов мањи део. Као и у случају IP датаграма у пољу опције може бити један или више бајтова променљиве дужине приказане на слици 3.9.

---

[1] *Checksum*
[2] За TCP протокол ово поље има вредност 6.
[3] *Urgent Pointer*
[4] *Options*
[5] Онолико велики колико дозвољавају станице који учествују у размени података.
[6] Додатном анализом [Stivenson] се показује се да није увек боље пренеосити веће сегменте.

Слика 3.9 TCP-IP поље опције променљиве дужине

Тренутно је дефинисано седам опција[1] (табела 3.2)

Врста опције	Дужина	Значење
0	-	Крај листе опција
1	-	Нема активности
2	4	Максимална величина сегмента
3	3	Опсег величине прозора
4	2	Селективна потврда се дозвољава
5	x	Селективна потврда
8	10	Временска ознака

Табела 3.2 Опције у TCP заглављу

Максимална величина сегмента (MSS[2]) постоји само у сегменту типа[3] *Syn*, односно користи се само за време успостављања везе. Означава максималну дозвољену величину сегмента коју пријемна страна може да прихвати. Уколико један крај везе не добије податак о максималној величини сегмента од друге стране везе он онда претпоставља подразумевану величину од 536 бајтова. Максимална величина сегмента не мора да буде иста у оба правца везе. Уколико се опција (слика 3.10) не користи могућа је било која величина сегмента. Свака рачунарска мрежа има своју максималну јединицу за пренос (MTU[4]) и сваки сегмент се мора уклопити у њу[5]. Уколико се сегмент преноси мрежом чија је максимална јединица за пренос мања од величине тог сегмента гранични рутер га дели на два или више мањих сегмената.

Слика 3.10 Опција максимална величина сегмента

[1] Описане су у документу RFC1072.
[2] *Maximum Segment Size*
[3] Сегменте са постављеним SYN битом означавамо са *Syn*.
[4] *Maximum Transfer Unit*
[5] На пример: за локалне рачунарске мреже Етернет типа је 1460 бајтова, за локалне рачунарске мреже по IEEE802.3 је 1452 бајтова, а за неке BSD имплементације 1024 бајтова пошто захтевају мултипл од 512 бајтова.

Опсег величине прозора (WSO[1]) постоји само у *Syn* сегменту. Обе стране морају да пошаљу WSO опцију у свом *Syn* сегменту да би се омогућила промена величине прозора. У току трајања везе није га могуће мењати. Поље „величина прозора" је број који указује колико бајтова је пријемник спреман да прихвати. У случају да се користи опција[2] „опсег величине прозора", онда се вредност из поља „величина прозора" множи са 2^F, где је F померај, такав да је $2^F < 2^{14}$.

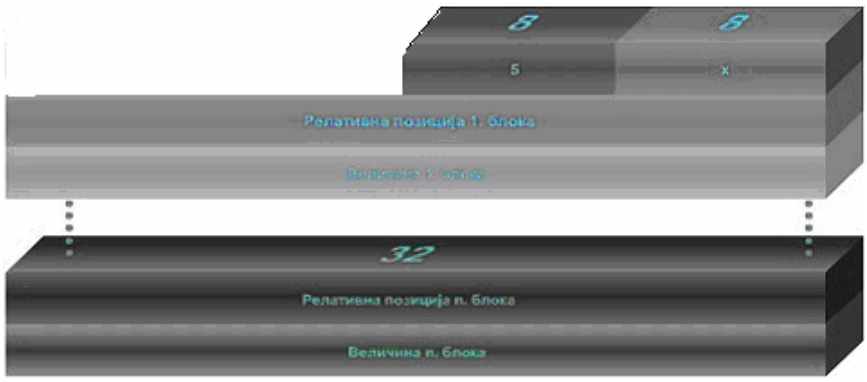
Слика 3.11 Опција опсег величине прозора

Поље „селективна потврда се дозвољава" (SACKP[3]) указује да је дозвољена употреба селективних потврда о успешном пријему података.

Слика 3.12 Опција селективна потврда се дозвољава

Поље „селективна потврда" (SACK[4]) омогућава примаоцу да обавести пошиљаоца о сегментима који су успешно примљени. Пошиљаоц ће поново послати само сегменте који нису успешно примљени[5]. Уколико је број сегмената које треба поново послати замашан дужина SACK поља била би превише велика. Због тога се на пријему подаци групишу у целине успешно примљених сегмената. Предајној страни се шаље информација о релативној позицији и величини целине. Обавештење о броју целина за које се може слати SACK опцијом је ограничен.

[1] *Window Scale Option*
[2] Дефинисана у документу RFC1323.
[3] *Selective Acknowledgeents Permitted (*SACK *Permitted*)
[4] *Selective* ACK
[5] Опције SACK и SACK *Permitted* дефинисане су у документу RFC2018.



Слика 3.13 Опција селективна потврда

Временска ознака (TS[1]) је поље које омогућава слање временских ознака у TCP сегментима[2]. Ова опција садржи два поља величине четири бајта. Поље „величина временске ознаке" (TSval[3]) садржи текући податак о такту предајне TCP целине. Садржај поља „ехо временске ознаке" (TSecr[4]) има значаја само уколико је постављен бит ACK у TCP заглављу.

Слика 3.14 Опција временске ознаке

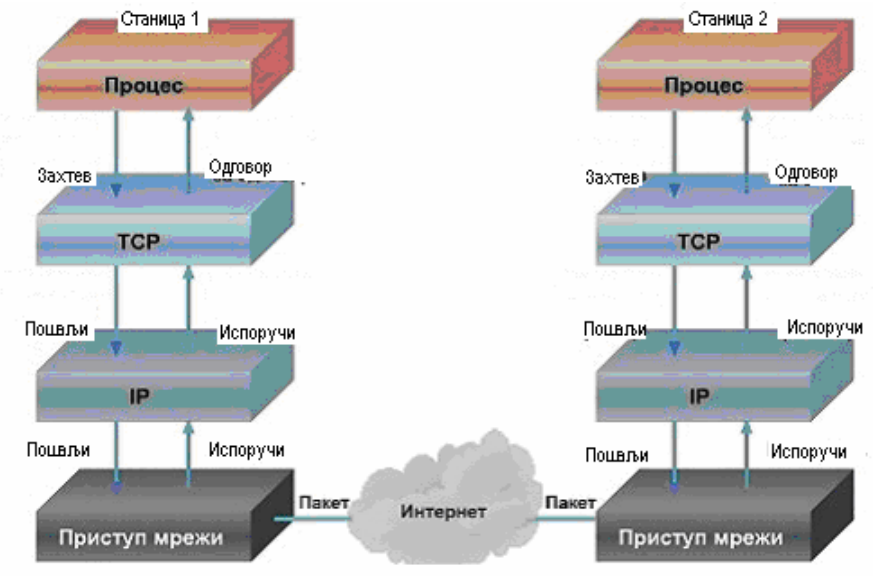
Употреба опције са временском ознаком омогућава предајнику да у сваки сегмент постави временску ознаку. Пријемник поставља исту вредност у *Ack* сегмент. Када предајна целина прими *Ack* сегмент она може да израчуна укупно време које је потребно да се пошаље пакет и да стигне његова потврда (RTT[5]).

3.5 Карактеристике TCP везе

Као што је претходним одељцима поменуто за сваки од слојева везан је одговарајући скуп примитива и параметара који имају за циљ пренос података и управљачких информације. Слика 3.15 показује контекст у коме се примитиве TCP услуга користе. Корисници на апликационом слоју размењују са TCP целином примитиве „Захтев" и

[1] *Timestamp*
[2] Ова опција је дефинисана у документу RFC1323.
[3] *Timestamp value*
[4] *Timestamp Echo Reply*
[5] *Round Trip Time*

„Одговор". Многе од њих иницирају слање једног или више сегмената ка удаљеној прикључници (сокету). Ове сегменте прихвата мрежни слој (IP) преко својих примитива „Пошаљи"и дистрибуира их одредишној TCP целини преко примитиве „Испоручи".



Слика 3.15 Примитиве које се размењују код услуга TCP, IP и приступа мрежи

Успостављање везе

За TCP везу потребна је трострука размена управљачких сегмената. Када је на једној страни везе клијент а на другој страни сервер тада да би се остварила веза:

- једна страна, сервер, пасивно чека на долазећу везу извршавајући Passiv Open примитиву, указујући (или не указујући) на одређено извориште;
- страна која упућује захтев (клијент) извршава Activ Open примитиву одређујући IP адресу и порт са којим жели да се повеже, максималну величину TCP сегмента коју је спремна да прихвати и опционо неке корисничке податке (нпр. шифру). Activ Open примитива доводи до слања TCP сегмента (*Syn* сегмент) са ознакама постављеним на следеће вредности: SYN=1 и ACK=0, број порта сервера са којим жели да се повеже и клијентов почетни (иницијални) редни број (ISS[1]). На слици 3.16 овај сегмент означен је као сегмент 1;
- када овај сегмент стигне на одредиште TCP целина проверава да ли постоји процес који је поставио Passiv Open примитиву на порт који је уписан у пољу за

[1] *Initial Send Sequence*

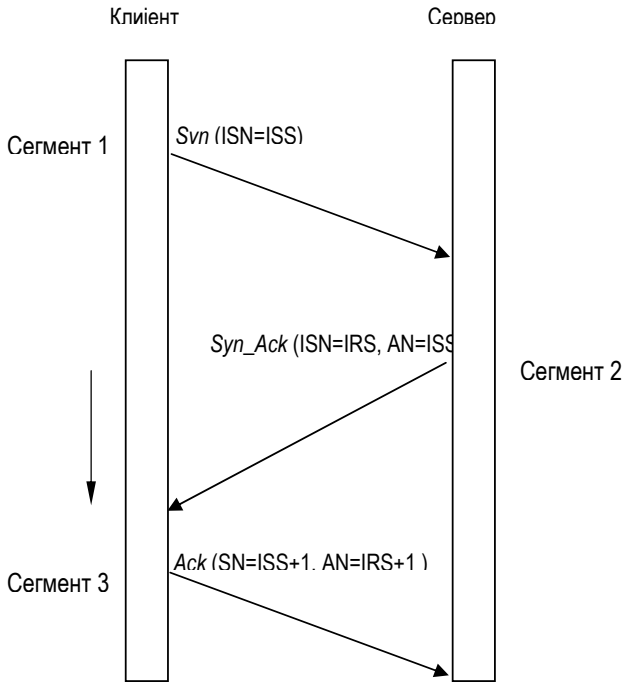
одредишни порт примљеног сегмента. Уколико постоји процес који „ослушкује" на специфицираном порту њему се прослеђује долазећи TCP сегмент. Ако порт прихвати везу шаље свој TCP сегмент (*Syn_Ack* сегмент) са ознакама постављеним на следеће вредности: SYN=1 и ACK=1. Сервер потврђује редни број SYN сегмента клијента постављањем потврдног редног броја на почетни редни број клијента увећан за један (ISS+1), серверов почетни (иницијални) редни број (IRS[1]). На слици 3.16 овај сегмент означен је као сегмент 2;

- клијент мора да одговори на пријем овог *Syn_Ack* сегмента слањем *Ack* сегмента са вредностима: SYN=0, ACK=1, потврдним редним бројем који је једнак почетној вредности редног броја сервера увећаног за један (IRS+1). На слици 3.16 овај сегмент означен је као сегмент 3;
- редослед TCP сегмената који се обично шаљу приказан је на слици 3.16.

Као што смо видели свака од страна поставља свој почетни (иницијални) редни број за ту везу (ISS, IRS). Почетни редни број мора да се мења у току времена, тако да свака веза има различит. У документу RFC739 специфицирано је да иницијални редни број треба посматрати као бројач са 32 бита који се инкрементира сваких 4μsec[2]. Намена овог броја је да се сегменти који пристижу са великим кашњењем погрешно не интерпретирају као део постојеће везе.

[1] *Initial Receive Sequence*

[2] Већина имплентација које су проистекле из Беркли дистрибуције BSD (*Berkly Software Distribution*) користи 8μsec.



Слика 3.16 Временски редослед размене сегмената код успостављања везе

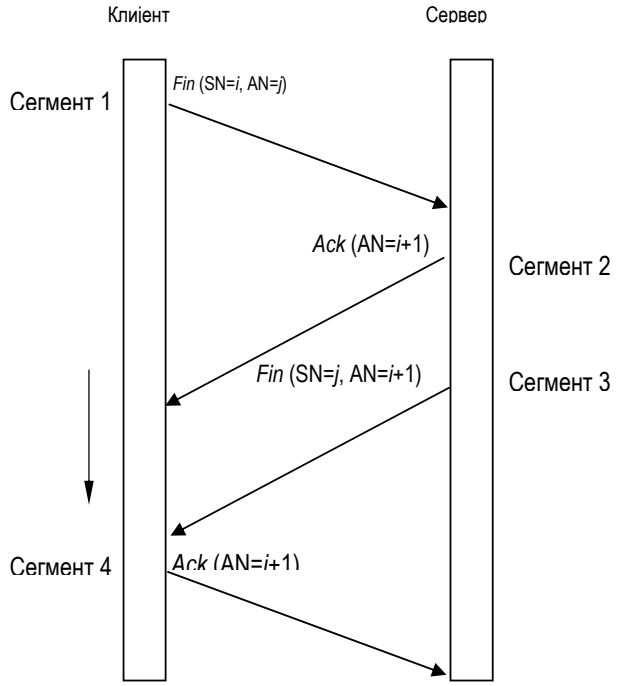
Постоје ситуације када веза не може да се успостави. Један од примера био би да сервер није подигнут, или да је кабл којим је клијент повезан на локалну рачунарску мрежу у прекиду. Питање је колико често ће TCP клијент слати *Syn* сегмент да би успоставио везу. На пример други *Syn* сегмент биће послати 5,8sec после слања првог *Syn* сегмента а трећи 24sec после другог сегмента. Колико дуго ће клијент покушавати да поново шаље *Syn* сегмент пре него што одустане? Већина система заснованих на Беркли имплементацији поставља временско ограничење на 75sec.

Раскидање TCP везе

За успостављање TCP везе потребно је да клијент и сервер размене три сегмента док је за раскид везе потребно да размене четири сегмента. Разлог је што TCP веза може бити и полузатворена.

Као што је познато TCP веза је истовремено двосмерна (потпуни дуплекс). Да би се боље објаснио раскид везе најбоље је посматрати TCP везу као две једносмерне (симплекс) везе. Свака једносмерна веза раскида се независно. Да би прекинула везу било која страна може послати TCP сегмент са постављеним битом FIN на вредност 1 (*Fin* сегмент). То значи да нема више података за слање. По пријему *Fin* сегмента TCP целина мора да обавести апликацију да је друга страна завршила са слањем података. Апликација која хоће да затвори везу иницирала је слање *Fin* сегмента.

Подаци се у другом правцу могу и даље слати без икаквих ограничења. Ову могућност у пракси мали број TCP апликација користи. Нормални сценарио приказан је на слици 3.17. Када су оба правца затворена веза је раскинута.



Слика 3.17 Временски редослед размене сегмената код раскидања везе

Страна која иницира затварање везе (она која шаље први *Fin* сегмент) извршава активно затварање[1], а друга страна (која прима *Fin* сегмент) извршава пасивно затварање[2].

Сегмент 1 (сл. 3.17) који иницира крај везе шаље се на пример када Telnet клијент затвара везу откуцавши команду „*quite*". То доводи клијент страну TCP везе у ситуацију да пошаље *Fin* сегмент окончавајући ток података од клијента ка серверу.

Када сервер прими *Fin* сегмент он шаље натраг *Ack* сегмент и инкрементира предајни редни број за 1 (сегмент 2). *Fin* сегмент користи редне бројеве баш као и *Syn* сегмент. У овој тачци TCP процес на страни сервера прослеђује апликацији информацију о крају[3] датотеке. Сервер затим затвара везу налажући TCP целини да пошаље *Fin* сегмент

[1] Active Close
[2] Pasive Close
[3] *End of file* - EOF

(сегмент 3) који клијент мора да потврди *Ack* сегментом у коме је примљени редни број увећен за један (сегмент 4).

Треба напоменути да слање *Fin* сегмената изазивају апликације на крајевима TCP везе а потврде *Fin* сегмената (*Ack* сегменте) аутоматски генерише TCP софтвер.

На слици 3.17 могли бисмо да променимо ознаке и да леву страну која иницира затварање везе означимо као сервер а десну као клијент. Уобичајено је ипак да процес клијента (као интерактивни корисник) први иницира прекид.

Мало је вероватно да оба краја TCP везе пошаљу *Fin* сегменте истовремено. У том случају оба би била потврђена на уобичајени начин и веза раскинута. У суштини нема никакве разлике у томе да ли је веза раскинута истовремено од обе стране или не.

Дијаграм стања TCP везе

Одржавање TCP везе захтева памћење неколико променљивих. Сматраћемо да су ове променљиве смештене у запису о вези који се обележава са TCB[1]. Међу променљивама које су смештене у запису о вези су: број локалне и удаљене прикључнице, сигурност везе, приоритети у вези, показивачи на предајни и пријемни меморијски простор корисника, показивачи на ретрансмисионе листе и текући сегмент. Поред поменутих променљивих које се односе на предајне и пријемне редне бројеве у запису о вези такође се памте[2] и:

- редни бројеви послатих сегмената;
  - послати и не потврђени (SND.UNA),
  - шаље се као следећи (SND.NXT[3]),
  - послати отвор прозора (SND.WND[4]),
  - послати показивач хитних података (SND.UP[5]),
  - редни број сегмента који је искоришћен за последње иновирање прозора (SND.WL1[6]),
  - редни број потврђеног сегмента који је искоришћен за последње иновирање прозора (SND.WL2[7]),
  - иницијални предајни редни број (ISS[8]),
- редни бројеви примљених сегмената;

---

[1] *Transmission Control Block*
[2] Назив и ознаке променљивих су преузете из документа RFC793.
[3] *Send next*
[4] *Send window*
[5] *Send urgent pointer*
[6] *Segment sequence number used for last window update*
[7] *Segment acknowledgment number used for last window update*
[8] *Initial send sequence number*

- следећи (RCV.NXT[1]),
- пријемни прозор (RCV.WND[2]),
- пријемни показивач хитних података (RCV.UP[3]),
- иницијални пријемни редни број (IRS[4]).

Објаснили смо бројна правила у иницирању и окончавању везе. Ова правила могу се приказати на дијаграму стања[5] везе који су приказани на сликама 3.5 и 3.18.

Имена и опис 11 стања која су дефинисана у RFC793 дати су у табели 3.3. Стање CLOSED у ствари је имагинарно стање када не постоји TCB запис а самим тим ни веза. У сваком стању дефинисани су прихватљиви догађаји. Уколико се они појаве предузимају се одређене акције. У противном пријављује се грешка.

Стање	Опис
CLOSED	Указује да не постоји не постоји TCB запис а самим тим ни веза.
LISTEN	Представља ишчекивање (ослушкивање) захтева за успоставом везе од удаљеног TCP процеса или порта.
SYN–SENT	Представља стање ишчекивања одговарајућег захтева за успоставом везе после слања захтева за успоставом везе (*Syn* сегмента).
SYN–RCVD	Представља стање ишчекивања пријема потврде захтева за успоставом везе (*Ack* сегмента) после и слања и пријема захтева за успоставом везе.
ESTABLISHED	Представља успостављену везу. Подаци који се добијају могу се испоручити кориснику. Ово је нормално стање за фазу преноса података у вези.
FIN–WAIT1	Представља стање ишчекивања захтева за окончањем везе од удаљеног TCP процеса, или потврду на захтев за окончањем везе који је претходно послат.
FIN–WAIT2	Представља ишчекивање захтева за окончањем везе од удаљеног TCP процеса.
TIME–WAIT	Представља чекање да протекне довољно времена да би били сигурни да је удаљени TCP процес примио потврду свог захтева за окончањем везе.

[1] *Receive next*
[2] *Receive window*
[3] *Receive urgent pointer*
[4] *Initial receive sequence number*
[5] *Finite state machine*

CLOSING	Представља ишчекивање потврде захтева за окончањем везе.
CLOSE–WAIT	Представља ишчекивање захтева за окончањем везе од локалног корисника.
LAST–ACK	Представља ишчекивање потврде захтева за окончањем везе претходно послатог удаљеном TCP процесу (што укључује потврду свих његових захтева за окончање везе).

Табела 3.3 Стања везе и њихов опис

Свака веза започиње стањем CLOSED. Из тог стања може се прећи у пасивно отворено LISTEN, или активно отворено CONNECT стање. Уколико друга страна има супротан прелаз веза се остварује и прелази се у ESTABLISHED стање. Раскид везе може да иницира било која страна. Када је он завршен прелази се у CLOSED стање.

Дијаграм стања приказан је на слици 3.18. Активно остваривање везе са пасивним сервером на слици је приказано пуном линијом која иде од клијента ка серверу. Пут пасивног отварања везе (сервер) приказан је на слици испрекиданом дебљом линијом. Танке линије представљају неуобичајени след догађаја. Свакој од линија придодат је пар догађај/активност[1].

Догађај може бити:

- системски позив иницијализован од стране корисника - примитиве Active Open, Pasive Open, Send, Close,
- долазећи сегмент *Syn*, *Fin*, *Ack* или *Rst*,
- истицање времена на који је постављен неки од часовника.

Активност је слање управљачког сегмента: *Syn*, *Fin*, *Ack* или *Rst*, или се ништа не предузима.

Дијаграм се најбоље анализира ако се прати пут клијента (дебља пуна линије) а затим пут сервера (испрекидане дебља линије). Када апликација на страни клијента тражи захтев за успоставом везе (Active Open примитива) локална TCP целина:

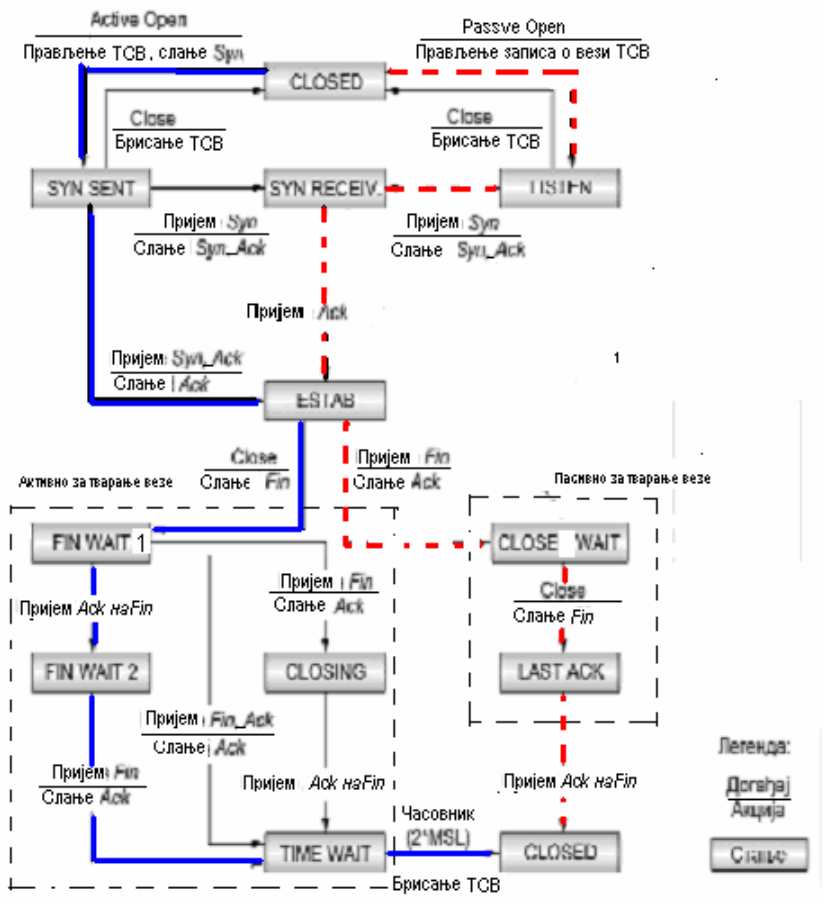
- прави запис о вези TCB,
- означава да је у стању SYN–SENT,
- шаље *Syn* сегмент.

Приметимо да више веза за више апликација може бити успостављено истовремено, тако да се стања односе на сваку поједину везу и уписују се у одговарајући запис о вези (TCB). Када стигне сегмент потврде[2] *Syn_Ack*, TCP целина шаље завршну потврду троструке

[1] *Event/action*

[2] То је сегмент код кога је: SYN=1, ACK=1.

размене и прелази у стање ESTABLISHED. Подаци се од тог тренутка надаље могу слати и примати.



Слика 3.18 Дијаграми прелаза из једног TCP стања у друго

Ако је апликација завршила са радом извршава Close примитиву која доводи до тога да локална TCP целина пошаље *Fin* сегмент, прелази у стање FIN–WAIT1 и чека потврду *Ack* на *Fin* сегмент (на слици 3.18 означено је испрекиданим квадром и обележено као активно затварње везе). Када стигне *Ack* на *Fin* сегмент прелази се у стање FIN–WAIT2 и један правац везе је сада затворен. Док је у стању FIN–WAIT2 клијент очекује још један сегмент - *Fin* сегмент који указује да и апликација на страни сервера окончава везу. По пријему тог *Fin* сегмента клијент га потврђује *Ack* сегментом и прелази у стање

TIME–WAIT. Док је у овом стању клијент може да поново пошаље *Ack* сегмент уколико је претходно послати изгубљен.

Сада су обе стране затвориле везу и TCP чека да истекне одређено време. Након истека овог времена TCP брише запис о вези.

Сада ћемо проучити управљање везом с тачке гледишта сервера. Сервер поставља Active Open и чека. Када стигне *Syn* сегмент, он бива потврђен и сервер прелази у стање SYN–RCVD. Када је *Syn* сегмент сервера потврђен завршава се трострука размена и сервер прелази у стање ESTABLISHED. Може да започне пренос података.

Ако клијент одлучи да прекине везу шаље Close примитиву која проузрокује слање *Fin* сегмента серверу (на слици 3.18 означено је испрекиданим квадром и обележено као пасивни прекид везе). По пријему TCP процес на страни сервера обавештава сервер апликацију. Када сервер апликација пошаље Close примитиву, TCP процес на страни сервера шаље клијенту *Fin* сегмент. Када стигне потврда клијента, сервер ослобађа везу и брише запис о вези (TCB[1]).

Стање чекања

Стање TIME–WAIT означава се и као стање чекања једнако двоструком трајању сегмента 2MSL[2]. Максимално трајање сегмента дефинише се као време постојања сегмента на мрежи. После истека тог времена сегмент се одбацује. Познато је да је ово време ограничено пошто се TCP сегменти преносе IP пакетима који у свом заглављу имају TTL поље које ограничава време постојања IP пакета[3]. За постављено 2MSL време важи следеће правило: када TCP целина извршава примитиву активног затварања везе и шаље последњи *Ack* сегмент та веза мора да остане у стању TIME–WAIT у трајању једнаком двострукој вредности времена MSL. Ово омогућава TCP процесу да ако треба пошаље последњи *Ack* сегмент за случај да се претходно послати изгубио (на удаљеној страни часовник ће сигнализирати да је време истекло и она ће поново да пошаље свој *Fin* сегмент).

Последица 2MSL чекања је и та што се пар прикључака[4] који дефинишу ту везу не може користити[5] док је TCP веза у стању 2MSL чекања. Тек по истеку времена једнаком 2MSL тај пар се може поново употребити[6].

Било који закаснели сегмент који стигне док је веза у 2MSL ишчекивању одбацује се пошто се веза дефинисана паром прикључака (која је у 2MSL ишчекивању) не може поново користити у том периоду. Када се успостави важећа веза са паром прикључака

---

[1] *Transmission Control Block*
[2] *Maximum Segment Lifetime* - максимално време живота сегмента.
[3] Подсетимо се да TTL (*Time To Live*) поље практично ограничава број рутера преко којих IP пакет прелаза а не време колико је IP пакет присутан у мрежи (тј. постоји).
[4] Сокет (*socket*) чине IP адреса клијента, број порта клијента, IP адреса сервера, број порта сервера.
[5] По дефиницији број локалног порта не може се поново употребити док год је тај број порта број локалног порта прикључнице која је у стању 2MSL ишчекивања.
[6] Неке имплементације имају начин да заобиђу ово ограничење.

који су претходно били у 2MSL ишчекивању а пристигну закаснели пакети из претходне везе[1] они неће бити погрешно интерпретирани.

Као што се види на слици 3.18 уобичајено је да клијент иницира активно затварање везе и улази у TIME–WAIT стање. Сервер обично обавља пасивно затварање везе и не пролази кроз TIME–WAIT стање. Последица је да ако клијент прекине са радом и исти клијент убрзо поново започне са радом онда он не сме поново да узме исти број локалног порта. За клијента то није никакв проблем пошто је за њега неважно које бројеве портова користи.

За сервер то није случај. Сервер користи познате бројеве портова. Уколико прекинемо рад сервера који је већ успоставио везу и убрзо покушамо поново да га подигнемо неће моћи да му се доделе претходно додељени број порта. Ти бројеви портова су део везе која је у стању чекања 2MSL. Може да протекне од 1 до 4 минута пре него што се сервер може поново покренути[2].

Неке имплементације дозвољавају постојање нове везе и за време TIME-WAIT стања уколико је нови иницијални редни број довољно већи од последњег редног броја претходне везе са истим локалним и удаљеним IP адресам и бројевима портова[3].

Концепт времена мировања[4]

Ишчекивање 2MLS обезбеђује заштиту од интерпретације закаснелих сегмената из претходне везе са истим истим локалним и удаљеним IP адресам и бројевима портова као сегменти текуће везе. Али ово функционише само под условом да не дође до прекида у раду станице која је у ишчекивању да протекне 2MLS.

Шта се дешава ако је у квару станица са портовима који су у чекању 2MLS? Поново се подиже после MLS секунди и одмах успоставља везу са истим локалним и удаљеним IP адресам и бројевима портова као и оним који су у чекању 2MLS. У том случају сегменти који су у кашњењу припадали вези пре квара система могу се погрешно интерпретирати да припадају новој вези креираној после поновног подизања система. Ово се може десити независно од тога који је иницијални редни број употребљен после подизања система.

Да би се заштитили од овакве грешке у документу RFC793 је предложено да TCP процес првих MLS секунди после подизања не треба да формира ни једну везу. Ово време назива се време мировања.

Стање FIN–WAIT2

У стању FIN–WAIT2 извориште шаље сегмент *Fin* и одредиште га потврђује. Ако није урађено полузатварање везе чека се на апликацију са другог краја везе да препозна да је

---

[1] Веза се дефинише паром прикључака (сокета). Нова веза са истим паром прикључака назива се реинкарнација те везе.
[2] У литератури [Куросе] се наводи да је ово време од 0,5, 1 и 2 минута.
[3] RFC1185 указује на проблеме који и на овај начин могу наступити.
[4] *Quite Time*

добила EOF[1] обавештење и да она затвара свој крај везе (Close) што доводи до слања сегмента *Fin*. Само када одредиште уради затварање везе изворишни крај се помера из стања FIN–WAIT2 у стање TIME–WAIT.

Ово значи да изворишни крај везе може да остане у овом стању у недоглед. Друга страна је и даље у CLOSE–WAIT стању и може у њему да остане заувек, односно све док апликација не затражи затварање везе.

Многе апликације засноване на Беркли имплементацији спречавају ово могуће бесконачно чекање у стању FIN–WAIT2 на следећи начин: уколико апликација иницира активно затварање везе онда она извршава потпуно затварање, а не полузатварање, указујући да очекује пријем података а затим активира часовник. Уколико је веза неактивна 10min плус 75sec, TCP процес помера везу у стање CLOSED. Овакав приступ није у сагласности са спецификацијом протокола.

Захтев за успоставу везе са непостојећим портом

Споменули смо да је део TCP заглавља и бит RST[2] (слика 3.7). У општем случају *Reset* се шаље увек када стигне сегмент који не одговара тој вези.

Уобичајен случај генерисања сегмента *Reset* је када стигне захтев за успоставу везе а не постоји процес који „ослушкује" на одредишном порту. Када је у питању UDP протокол онда се шаље ICM порука „одредишни порт је недоступан". TCP протокол уместо тога користи *Reset*.

Нагло прекидање везе

Видели смо да је нормалан начин да се веза оконча да једна од страна пошаље *Fin* сегмент. Овај начин некада се среће под називом нормално ослобађање (прекид) везе. Сегмент *Fin* се шаље тек када се претходно сви подаци који су били спремни за слање (смештени у реду за чекање) пошаљу. На тај начин не постоји могућност губитка података.

Могуће је окончати везу слањем сегмента *Reset* уместо *Fin* сегмента. То се назива нагло прекидање везе[3]. Нагли прекид везе изазивају две ствари:

- било који подаци који су у реду чекања за слање одмах се одбацују,
- пријемник сегмента *Reset* зна да је удаљена страна нагло прекинула везу.
- API[4] који користи апликација мора да обезбеди могућност наглог прекида уместо нормалног прекида везе.

Откривање полузатворене везе

[1] *End Of File*
[2] *Reset*
[3] *Abortive release*
[4] *APplication Interface*

TCP се налази у полузатвореном стању уколико је један крај везе затворио (нормално или нагло) везу без обавештавања друге стране о томе. Ово може да се деси када дође до квара на било којој страни[1] везе. Уколико не постоји покушај стране која је у квару да пошаље податке преко полуотворене везе страна која је још увек у отвореној вези неће открити да је друга страна у квару.

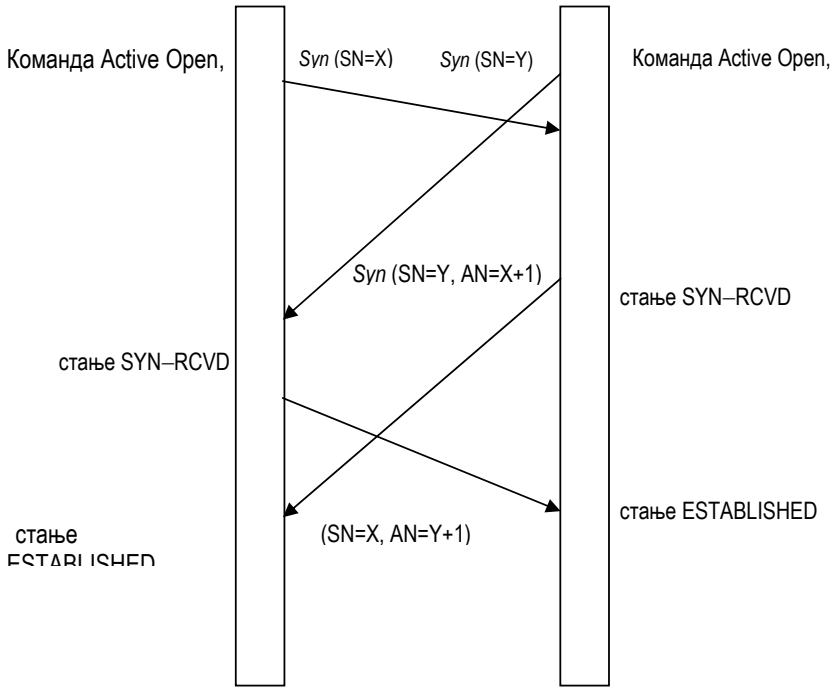
Други уобичајен узрок полуотворене везе је када станици на страни клијента искључи напајање уместо да оконча клијентску апликацију а тек затим регуларно искључи (обори) систем.

Ово се на пример дешава када је на персоналном рачунару покренут Telnet клијент, а корисник искључи рачунар на крају дана. Уколико у тренутку искључивања рачунара није био у току пренос података сервер никада неће сазнати да клијент није више на другој страни везе. Када се корисник појави следећег јутра, укључи свој рачунар, покрене новог Telnet клијента и отвори нову везу са сервером то може да доведе до превише полуотворених веза на серверу[2].

Истовремено отварање везе

Могуће је, иако мало вероватно, да апликације на оба краја изврше истовремено једна ка другој команду Active Open. Сваки крај мора да пошаље свој *Sin* сегмент. Такође се захтева да је сваком од крајева везе познат локални порт другог краја везе. Ово се назива истовремено отварање[3].

[1] Ако је веза успостављена између сервера или клијента онда било сервер било клијент. Ако је веза успостављена између две станице било која од њих.
[2] Постоји начин да један крај TCP везе открије да је други нестао уз помоћ опције „TCP *keepalive*".
[3] *Simultaneous Open*



Слика 3.19 Размена сегмената за време истовременог отварање везе

На пример једна апликација на страни станице 2 може да има локални порт 6000 и извршава команду Active Open са бројем порта 8000 на станици 1. Апликација на станици 1 има као број локалног порта 8000 и извршаваће команду Active Open са бројем порта 6000 на станици 2.

Ово није исто као када се истовремено повезују Telnet клијент на станици 1 са Telnet сервером на станици 2 и Telnet клијент на станици 2 са Telnet сервером на станици 1. У овом случају оба Telnet сервера извршавају команду Passive Open а не команду Active Open. Telnet клијенти додељују себи пролазне (ефемерне) бројеве портова а не бројеве портова који су познати другим Telnet серверима.

TCP протокол је пројектован тако да може да разреши истовремено отварање везе. Правило је да се као резултат добија само једна а не две везе. Код неких других стандарда као што је на пример транспортни протокол код OSI референтног модела као резултат истовременог отварања би дао две отворене везе. Када дође до истовременог отварање везе прелаз из једног стања у друго је различит од оног на слици 3.18. Сценарио је следећи:

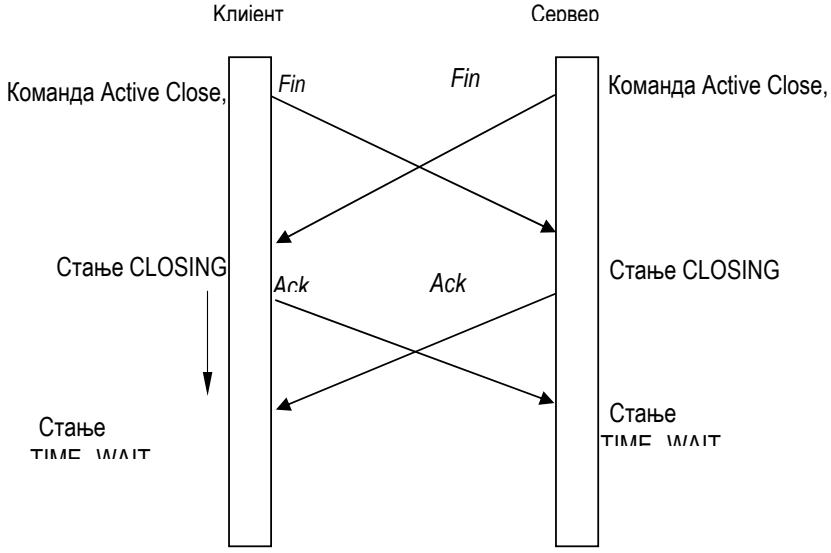
- оба краја шаљу *Syn* сегмент у исто време и улазе у SYN–SENT стање,

- када сваки од крајева прими *Syn* сегмент прелази у стање SYN–RCVD,
- поново шаљу *Syn* сегмент и потврђују примљени *Syn* сегмент из супротног правца (*Syn_Ack*),
- по пријему *Syn* сегмента са потврдом прелази се у стање ESTABLISHED.

Истовремено затварање везе

Споменули смо у претходом поглављу да једна страна (уобичајено је да је то клијент) извршава активно затварње везе проузрокујући слање *Fin* сегмента. Могуће је за обе стране да извршавају команду активног затварања и да се TCP протоколу дозвољава истовремено затварање везе.

Када апликације пошаљу примитиве Close (слика 3.18) TCP целине на страни сервера и клијента из стања ESTABLISHED прелазе у стање FIN–WAIT1и шаљу *Fin* сегменте. Сваки од крајева везе када прими *Fin* супротне стране прелази из стања FIN–WAIT1 у стање CLOSING и шаље завшни *Ack* сегмент. Кад сваки од TCP процеса на крајевима везе прими одговарајући *Ack* сегмент прелази у стање TIME–WAIT (слика 3.20). Са истовременим затварањем везе исти број сегмената се размењује као и у нормалном затварању везе.



Слика 3.19 Размена сегмената за време истовременог затварања везе

3. Транспортни слој на Интернету – други део

3.6 Методе у размени података

TCP стандард обезбеђује прецизне спецификације за TCP протокол који се користи за размену података између станица. У овом поглављу анализираћемо различите методе (стратегије). Они се могу категоризовати на следећи начин:

- метод за слање података[1],
- метод за испоруку података[2],
- метод за прихватање података[3],
- метод за поновно слање података[4],
- метод за потврде о пријему података [5].

Метод за слање података

TCP целина је слободна да изабере начин на који ће да шаље податке у границама кредита[6] који поседује у случају да се не ради о подацима које треба одмах испоручити[7]. Када корисник проследи податке TCP целини она их смешта у предајни меморијски простор. TCP целина може да направи сегмент од сваке појединачне групе података које јој корисник проследи или да сачека да се скупи одређена количина података пре него што направи сегмент (слика 3.5). Метод (начин) који одабира биће прилагођен тренутним захтевима везе. Ако је пренос података:

- велики и неравномеран постојаће мала количина додатних некорисних података[8] (премашење[9]),
- систем код кога се често шаљу сегменти мале величине биће бржи у обради података и одговору на њих.

Метод испоруке података

---

[1] *Send Policy*
[2] *Deliver Policy*
[3] *Accept Policy*
[4] *Retransmit Policy*
[5] *Acknowledge Policy*
[6] Отвора прозора.
[7] Са постављеним PSH битом на вредност 1.
[8] Под појмом корисни подаци подразумевају се сами подаци (*data, payload*) који се преносе. Подаци који нису корисни, тзв. некорисни односе се на заглавље.
[9] *Overhead*

Када се не користи функција за тренутно испоручивање података кориснику TCP целина може да испоручује податке кориснику према свом плану, односно:

- може да испоручује податке кориснику оним редом који они стижу или,
- може прво да прикупи одређену количину података у свој пријемни бафер па да их онда све заједно пошаље кориснику.

Који ће се метод користити зависи од тренутних захтева. Ако је интензитет преноса података неравномеран и ако се преноси велика количина података може се десити да корисник не добија податке довољно често како би било пожељно. С друге стране, ако је интензитет преноса равномеран и на тај начин се преносе мали сегменти, може да дође до непотребне обраде сваког сегмента код предајне и код пријемне стране, па и до непотребних честих прекида које пријемна TCP целина шаље кориснику да би преузео податке.

Метод за прихватање података

TCP целина прима податке оним редом којим су они стигли и смешта их у пријемни меморисјки простор да би их испоручила кориснику. Може да се деси да подаци не стигну редоследом којим су послати. У том случају TCP целина може да примени два метода:

- метод „по реду "[1] који омогућава прихватање само оних сегмената који стижу у правилном редоследу,
- метод „у оквиру прозора"[2] који омогућава прихватање сегмената који стижу у складу са тренутним отвором пријемног прозора.

Метод који обезбеђује прихватање само оних сегментата који стигну у правилном редоследу (на пример SN=1, SN=201, SN=401...) лако је применити. Последица примене овог метода је велики број ретрансмисија. Могуће је да се деси да неки сегменти не стигну на одредиште у право време (на пример, сегмент са редним бројем 201 може да стигне после сегмента са редним бројем 401). У том случају пријемна TCP целина одбацује тај сегмент. Када време за добијање потврде о пријему истекне предајна TCP целина поново шаље тај сегмент.

Значи да ако је само један сегмент стигао касније сви сегменти који стигну после њега бивају одбачени и предајник ће морати поново да их пошаље. Ово није добро решење јер циљ TCP протокола јесте да обезбеди што мање ретрансмисија.

Метод код ког се прихватају сви сегменти са редним бројевима, који су у оквиру пријемног прозора, много је прихватљивији. Разлог је једноставан: битно смањује потребу за ретрансмисијом. Мана овог метода је што је сложенији за реализацију. Мора се стално проверавати да ли примљени сегмент испуњава услов и мора се водити рачуна који су сегменти стигли изван правилног редоследа.

Метод за поновно слање података

---

[1] *In-order*
[2] *In-window*

TCP целина води рачуна о реду који чине сегменти који су послати али нису још потврђени. Када потврда за сегмент који је послат не стигне у одређеном времену[1] предајна TCP целина га поново шаље. Постоје три метода за поновно слање:

Слање само првог сегмента[2]. Постоји само један часовник додељен реду за ретрансмисију. Када стигне потврда (*Ack* сегмент) да је одређени сегмент успешно примљен онда се тај сегмент уклања из реда за ретрансмисију и часовник се поново покреће. У случају да време истекне пре него што стигне потврда о успешном пријему било ког сегмента из реда поново се шаље први сегмент из реда и часовник се поново покреће.

Слање групе сегмената[3]. Постоји само један часовник додељен реду за ретрансмисију. Када се прими потврда о пријему одређеног сегмента онда се тај сегмент уклања из реда за ретрансмисију и часовник се поново покреће. У случају да време на које је часовнику подешен истекне пре него што стигне потврда о пријему било ког сегмента из реда онда се сви сегменти из реда поново шаљу а часовник се поново покреће.

Слање појединачних сегмената[4]. Сваком сегменту који се налази у реду за ретрансмисију додељен је по један часовник. Када се за одређени сегмент прими потврда о пријему, онда се тај сегмент уклања из реда за ретрансмисију а њему додељен часовник зауставља. У случају да време за пријем потврде истекне на неком од часовника, онда се сегмент на који тај часовник указује поново шаље а часовник поново покреће.

Првоописани метод (слање само првог сегмента) ефикасан је по питању броја ретрансмисија, јер се поново шаљу само они сегменти за које није стигла потврда о пријему. Недостатак овог метода је тај што часовник за ретрансмисију указује само на први сегмент у реду. То значи да није могуће покренути часовник за сегмент који је други у реду док се не добије потврда о пријему за сегмент који је први у реду за ретрансмисију, чиме се уноси одређено кашњење. Метод са слањем појединачних сегмената разрешава овај проблем, али је доста комплекснији за имплементацију. Слање групе сегмената такође може да реши поменути проблем, али може и да створи нов - велики број сегмената који се непотребно поново шаљу.

Ефективност изабраног метода за ретрансмисију директно зависи од метода (стратегије) за пријем који користи пријемна страна. Ако пријемна страна користи:

метод редоследа онда ће она одбацити сегменте који су по реду иза изгубљеног сегмента. Тако је у овом случају боље на предајној страни применити метод са групом сегмената јер ће се на тај начин поново послати сви сегменти за које није стигла потврда, а међу њима и сегмент који је изгубљен;

---

[1] Време за добијање потврде о пријему одређеног сегмента истекне пошто је реч о одговарајућем часовнику који се поставља на одређену вредност.
[2] *First only*
[3] *Batch*
[4] *Individual*

метод отвора прозора онда је најбоље на предајној страни користити метод слања првог сегмента, или слање појединачних сегмената.

Метод за потврду пријема података

Када пријемна страна прими сегмент који је правилног редоследа[1] има две опције везане за метод (начин) како ће реаговати:

Тренутно слање потврде. Одмах шаље празну потврду (сегмента *Ack*) без података и са одговарајућим потврдним редним бројем (AN).

Кумулативно слање потврде. Када прими податак пријемна TCP целина запамти да је потребно послати потврду. Потврду не шаље одмах већ ће сачекати први свој сегмент са подацима који се шаље другој страни. Редни број потврде биће уписан у заглавље тог сегмента[2]. Да би се спречила дугачка кашњења треба активирати часовник за отвор прозора (табела 3.1). Уколико часовник сигнализира истицање времена на које је подешен пре него што је *Ack* сегмент послат треба послати празан сегмент који садржи одговарајући број сегмента.

Метод тренутног слања потврде једноставан је и обезбеђује да је удаљена TCP целина потпуно информисана чиме се ограничавају беспотребне ретрансмисије. Овај метод доводи до слања додатних празних, искључиво *Ack* сегмената. Такође може да изазове додатно повећање саобраћаја и самим тим и оптерећење мреже. Замислимо да TCP целина прими сегмент и одмах пошаље потврду (*Ack*). Подаци се одмах прослеђују апликацији, што обично води повећању пријемног прозора, изазивајући слање још једног празног сегмента који даје додатни кредит предајној TCP целини.

Због могућег преоптерећења обично се користи кумулативни метод. Треба уочити да је кумулативни метод сложенији за реализацију и израчунавање укупног времена потребног да се пренесе податак и да стигне потврда за њега (RTT[3]).

3.7 Интерактивни ток података

Анализа TCP саобраћаја показује да се он састојати од две врсте сегмената:

TCP сегмент који садржи велику количину[4] података који се преносе код апликација као што је пренос датотека FTP, електронска пошта SMTP и сл.

TCP сегмент који садржи интерактивне податке који се срећу код апликација Telnet , X-Windows, Rlogin и сл.

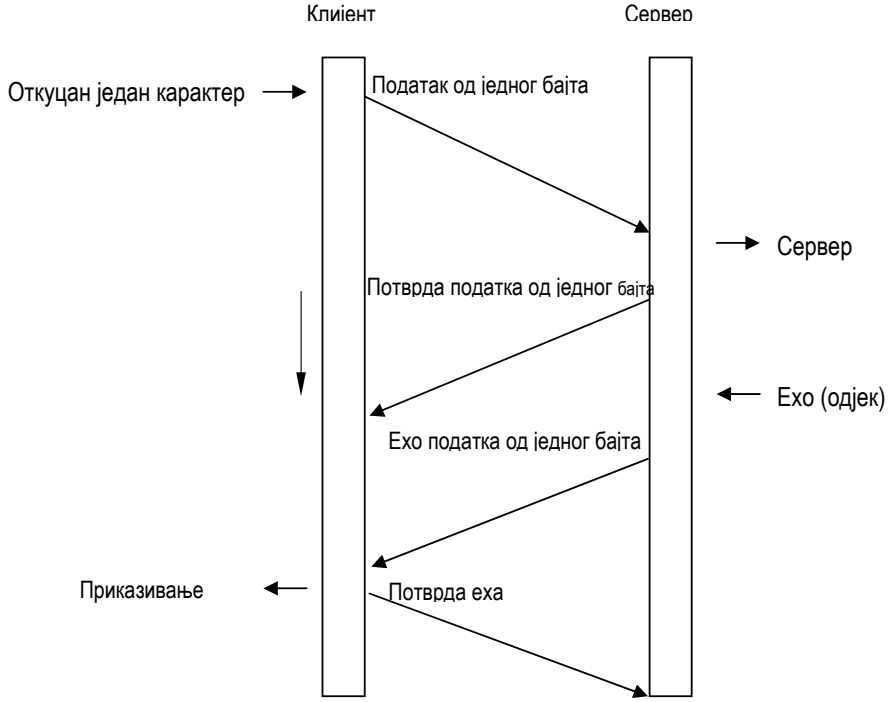
Ако би се посматрала заступљеност ове две врсте сегмената показује се да 90% саобраћаја чине TCP сегменти који носе велику количину података и обично су максимално могуће величине (бар 512 бајтова корисничких података). Само 10% TCP

[1] Правилан редослед подразумева да пакети стижу истим редоследом којим су и послати, односно са редним бројевима који су сукцесивни.
[2] Као што је у претходним поглављима већ поменуто овај мехнизам се на енг. означава са *piggy back*.
[3] *Round Trip Time*
[4] *Bulk*

сегмената су они који носе интерактивне корисничке податке и обично значајно мање величине[1].

Као пример узећемо Telnet везу са интерактивним едитором који реагује на сваки тастер. Размотрићемо метод најгорег случаја када TCP целина пошиљаоца прими један карактер, формира сегмент дужине 21 бајт[2] и прослеђује га IP целини. IP целина додаје своје заглавље и шаље га као IP пакет дужине 41 бајт[3]. Пријемник одмах шаље потврду дужине 40 бајтова. Касније, када едитор прочита примљени бајт, TCP целина шаље ажурирану величину отвора прозора померајући отвор прозора један бајт удесно. Овај пакет је такође дужине 40 бајтова. Коначно, након што едитор обради карактер шаље одјек (ехо) карактера као пакет дужине 41 бајт. Види се да су за сваки откуцани карактер употребљена 162 бајта и послата четири сегмента. Уколико је пропусни опсег система мали овај метод је неприхватљив.

[1] Поменуте интерактивне апликације као што су нпр. Telnet, Rlogin носе у просеку по 10 бајтова корисничких података [Stivenson].
[2] Податак је дужине 1 бајт (слика 3.20) и заглавље TCP сегмента је 20 бајтова.
[3] Заглавље IP датаграма је 20 бајтова. Заједно са упакованим (енкапсулираним) TCP сегментом је укупно 20+21=41. Премашење је 4000%.



Слика 3.20 Један могући начин за размену сегмената код интерактивне апликације

Један од метода за побољшање ефикасности пошиљаоца познат је као Неиглов[1] алгоритам[2]. Неигл је предложио једноставан метод: када подаци стижу до пошиљаоца бајт по бајт, он шаље само први бајт, а све преостале бајтове складишти док не стигне потврда за бајт који је послат. Затим шаље све ускладиштене карактере у једном TCP сегменту и започиње складиштење поново све док сви послати карактери не буду потврђени. Предност овог метода је у томе што је самотактујући. Што потврде брже стижу карактери се брже шаљу. Када се сегменти шаљу преко споре рачунарске мреже ширих подручја[3] корисно је смањио број малих сегмената[4] који су присутни у мрежи. Када се користи Етернет локална рачунарска мрежа онда је 16ms средње време које је потребно да се: пошаље један бајт, добије његова потврда и његов ехо[5]. Да би генерисао податке веће брзине од тог времена корисник би требало да откуца више од 60 карактера у

[1] *Nagle*
[2] Документ RFC896 издат 1984. год.
[3] WAN (*Wide Area Network*)
[4] *Tiny*
[5] Литература[Stivenson]

секунди што је мало вероватно. Ово значи да се Неиглов алгоритам ређе користи у локалним рачунарским мрежама[1].

Алгоритам додатно омогућава слање новог пакета уколико пристигне довољно података да би се попунила половина отвора прозора или максималан сегмент.

Неиглов алгоритам се доста користи али није најбоље решење за неке апликације. На пример код X-Windows апликација (које су значајно заступљене на Интернету) покрети миша треба да се пошаљу ка удаљеном рачунару. Сакупљање ових покрета и слање у групи (одједном) чини кретање миша неправилним.

3.7 Пренос велике количине података

Код преноса велике количине података[2] TCP протокол обезбеђује контролу тока са применом механизма клизајућег прозора[3] (слика 3.1). Дозвољава се пошиљаоцу да пошаље групу сегмената а и да очекује потврду за послату групу сегмената[4]. Наравно пренос је значајно убрзан у односу на метод заустави се и чекај[5].

Пријемни процес обично може контролисати величину прозора коју дозвољава. Величина прозора[6] клијента и сервера може да буде 2048, 4096[7], 8192 или чак 16384 бајта у зависности од верзија оперативног система. Величина пријемног меморијског простора једнака је максималној величини пријемног прозора за ту везу.

За Етернет локалне рачунарске мреже бафер величине 4096 бајтова (и на предајној и на пријемној страни) није оптималан. Пропусна моћ система порасла је за 40% када је меморијски простор (бафер) повећан на 16384 бајтова[8].

Убрзана испорука на пријему

Функција са постављањем PUSH ознаке омогућава TCP корисницима да врше контролу својих меморијских простора.

Употреба меморијских простора повећава ефикасност тако што омогућава предајној TCP целина да може да шаље мање већих сегмената уместо више мањих. На тај начин је мањи број обрада сегмената и на страни пошиљаоца и на страни примаоца. Пријемна TCP целина смањује број захтева који се шаљу апликацији за преузимање података. Негативна страна употребе меморијског простора је што доводи до повећања кашњење у испоруци података.

---

[1] Постоји могућност да се он по потреби искључи.
[2] *Bulk data flow*
[3] *Sliding Window*
[4] Детаљно је објашњено у претходним поглављима
[5] Метод „заустави се и чекај" (*Stop and Wait*) објашњен је 11. глави литературе [1]. Користи га једноставан транспортни протокол за пренос датотека TFTP (*Trivial File Transport Protocol*).
[6] Користи се термин објављена величина прозора (*advertised widow size*)у литератури [Stivenson]
[7] Сматра се подразумеваном (*default*) вредношћу.
[8] [Papadopulos, Parulkar]

Ако се користи PUSH функција онда је апликацији омогућено да заобиђе складиштење података а тиме се и кашњење смањује.

У оригиналној TCP спецификацији предвиђено је да апликација може да утиче на TCP целину да постави PSH бит на вредност један. У интерактивним апликацијама када клијент шаље команду серверу клијент ће поставити бит PSH=1. Дозвољавање клијентској апликацији да наложи TCP целини постављање PUSH бита служи да обавести клијентску TCP целину да процес на страни клијента не жели да се подаци задржавају у меморијском простору пријемника. Када на страни сервера TCP целина прихвати сегмент са постављеним PUSH ознаком то је њој обавештење да одмах пошаље податке апликацији а не да чека да ли ће пристићи још неки подаци.

Многе данашње апликације не подржавају могућност постављања PSH ознаке. Добро конципиран TCP процес сам процењује када треба да постави PSH ознаку. Већи број апликације изведене из Беркли имплементације аутоматски постављају PSH ознаку уколико послати сегмент празни предајни меморијски простор. На пријему ту PSH ознаку игноришу пошто никада не касне са испоруком примљених података (одмах их испоручују).

Хитна испорука

TCP целина обезбеђује механизам хитне испоруке[1] који омогућава једном крају да обавести други крај да су „хитни" подаци смештени у нормалан ток података. На пријемнику је да одлучи шта ће и како да уради са тим подацима.

Обавештавање о постојању хитних података обавља се постављањем два поља у TCP заглављу (слика 3.7). Бит означен са URG постављен је на 1 и показивач хитних података је постављен на позитивну вредност (одступање), којој се додаје редни број у TCP заглављу да би се добио редни број последњег бајта хитних података.

TCP целина мора да обавести пријемни процес када стигне сегмент са показивачем хитних података. Апликација на пријему ће очитати податке и она мора да буде у стању да процени како да искористи показивач хитних података.

Сама TCP целина веома мало казује о хитним подацима. Не постоји начин да се покаже где је почетак хитних података у самом току података. Једина информација која се шаље кроз TCP везу је да се ради са хитним подацима (бит URG у TCP заглављу постављен на 1) и показивач последњег бајта хитних података. Све остало је препуштено самој апликацији.

Поставља се питање која је намена коришћења рада са хитним подацима? Две уобичајене апликације су Telnet и Rlogin и случај када интерактивни корисник притисне тастер за прекид. Други пример је FTP када интерактивни корисник прекида пренос датотека.

Telnet и Rlogin користе рад са хитним подацима од стране сервера ка клијенту пошто је могуће да тај правац података заустави клијентска страна TCP везе (огласи прозор 0). Уколико процес на страни сервера користи врсту рада са хитним подацима, TCP целина

[1] *Urgent mode*

на страни сервера одмах шаље показивач и ознаку хитних података иако не може да шаље никакве податке. Када TCP целина на страни клијента прими информацију, она као одговор обавештава процес на страни клијента тако да клијент може да чита са сервера, отвори прозор и дозволи ток података.

Шта се дешава ако пошиљалац неколико пута започиње рад са хитним подацима пре него што пријемник обради све податке закључно са првим показивачем? На пријемној страни губи се информација о претходној позицији показивача. Постоји само један показивач хитних података на пријему и новопристигли показивач се уписује преко претходног. Ово значи да уколико су подаци важни онда извориште мора специјално да их обележи. На пример Telnet све бајтове команди у току података обележава бајтом префикса чија је вредност 255.

Успорени почетак

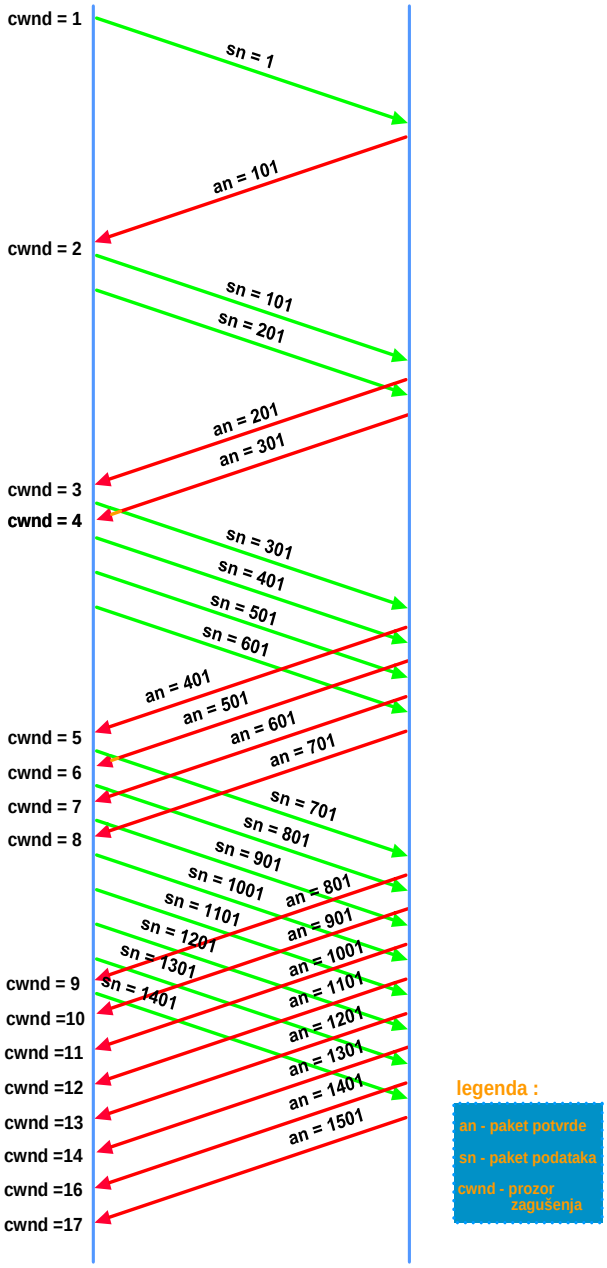
У претходним поглављима претпостављали смо да на самом почетку предајник шаље групу сегмената. Максимална величина групе одговара величини (објављеног) прозора пријемника. Када су станице на истој локалној рачунарској мрежи овакав начин рада је прихватљив. Уколико су предајна и пријемна станица повезане преко рутера а неке од међувеза су мање брзине може да дође до одређених проблема. На пример неки од рутера примљене пакете стављају у редове[1] за чекање. Уколико рутер не стигне да обради пакете из реда, тј. уколико се не ослободи простор у прихватном меморијском простору онда су новопристигли пакети изгубљени. На овај начин[2] може се значајно смањити пропусна моћ TCP везе.

Алгоритом под називом успорени почетак[3] који се захтева да TCP целине подржи има за циљ превазилажење поменутих проблема. Принцип на коме ради алгоритам је следећи: брзина којом се пакети убацују у мрежу одређена је брзином којом одредиште враћа потврде.

Спори почетак додаје још један прозор предајној TCP целини: прозор загушења[4]. После успоставе везе пошиљалац одређује величину прозора загушења на основу величине сегмента који је у тој вези у употреби[5] и шаље само један сегмент те величине. Ако потврда (*Ack* сегмент) стигне пре истека часовника загушења онда се прозор загушење увећава за величину још једног сегмента. Шаљу се два сегмента и чека потврда (слика 3..21). Када се сваки од ових сегмената потврди прозор загушења се увећава за четири. На овај начин обезбеђен је експоненцијални раст прозора загушења. У неком тренутку се достигне капацитет Интернета и неки од рутера почне да одбацује пакете. То је знак предајнику да је прозор загушења отишао предалеко и да се треба зауставити. На пример: ако количина од 1024, 2048 и 4096 бајтова пролази кроз мрежу без проблема али

[1] Редови за чекање су уствари резервисани меморисјки простор (бафер).
[2] [Jacobson 1988]
[3] *Slow start*
[4] *Congestion window*
[5] Односно максималне величине сегмента који је објављен у тој вези.

количина од 8192 бајта изазива кашњење прозор загушења биће подешен на 4096 бајтова да би се избегло загушење.



Слика 3.21 Пример успореног почетка

Колика је та максимална вредност изражена у бајтовима која може да се пошаље? Број бајтова који може да се пошаље одговара броју бајтова мањег прозора. То значи да је величина прозора једнака величини мањег од прозора: објављеног прозора (прозора примаоца) или прозора загушења (прозора пошиљаоца). Ако прималац захтева слање 8kB података, а пошиљалац зна да количина података већа од 4kB изазива преоптерећење мреже, шаље се 4kB. У другом случају ако прималац захтева слање 8kB а пошиљалац зна да количина до 32kB без проблема пролази кроз мрежу шаље се 8kB. Можемо да закључимо да:

- прозором загушења предајник реализује механизам контроле тока предајника,
- објављеним прозором пријемник реализује мехнизам контроле тока пријемника.

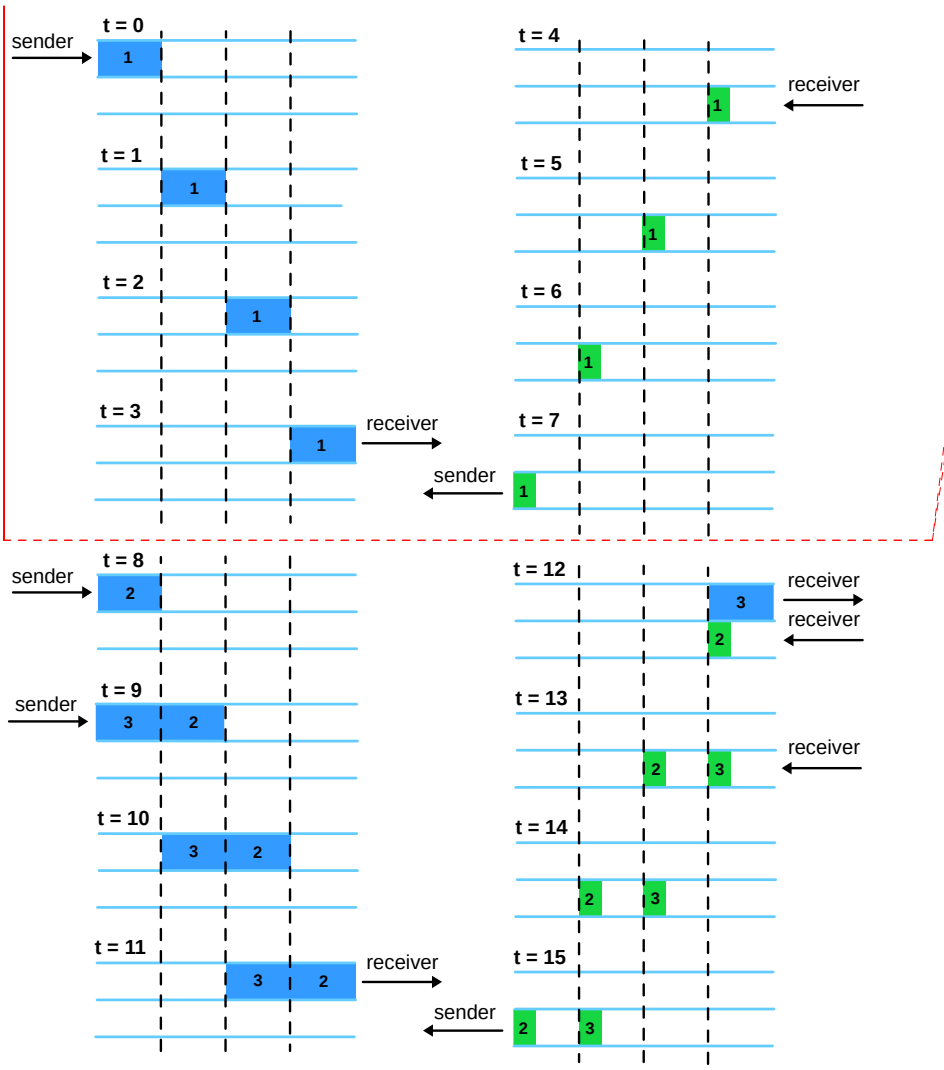
У једном од следећих поглавља видећемо како се успорени почетак уобичајено примењује уз метод који се назива заобилажење загушења[1].

Пропусна моћ код преноса велике количине података

У овом делу анализираћемо међусобни утицај величине прозора, контроле тока и успореног почетка.

Слика 3.22 приказује у времен поједине кораке између пошиљаоца и примаоца у. Приказано је шеснаест дискретних тренутака у времену. Сегменти који носе податке означени су са 1, 2, 3 итд. Потврде ових сегмената означене су са A_1, A_2, A_3 итд.

[1] *Congestion avoidance*



Comment [A1]: Page: 1
лика 20.8 Стивенсон на страни 287 и комбинација слика e-book 109 стр. 247 TCPIP tutorial. pdf

Comment [A2]: Page: 1
лика 20.8 Стивенсон на страни 287 и комбинација слика e-book 109 стр. 247 TCPIP tutorial. pdf

Слика 3.22 Временски тренуци *t*=0 до *t*=15 у примеру преноса велике количине података

У тренутку *t*=0 предајна TCP целини шаље један сегмент. Пошто је у режиму „успорени почетак" (прозор загушења је величине једног сегмента) предајна TCP целини мора да сачека да стигне потврда за овај сегмент па тек онда да настави са слањем. У тренуцима *t*=1, *t*=2 и *t*=3 сегменти се померају за по једну временску јединицу удесно. У тренутку *t*=4 пријемник очитава сегмент и шаље потврду A_1. У тренуцима *t*=5, *t*=6 и *t*=7 потврда A_1 се помера улево за по једну временску јединицу. Добили смо да је време потребно да се пошаље сегмент и да за њега стигне потврда RTT= 8 временских јединица.

На слици су нацртани сегменти потврде мање величине од сегмената података пошто се они састоје само IP и TCP заглавља. Приказан је само пренос података у једном правцу. Такође смо претпоставили да је брзина преноса у оба правца иста (што није увек тачно). У општем случају време потребно да се пошаље пакет састоји се од:

- времена простирања (пропагације). Назива се и кашњење због простирања,
- времена потребног да се пошаље пакет које зависи од брзине преноса (колико битова у секунди медијум може да пренесе). Назива се и кашњења због преноса.

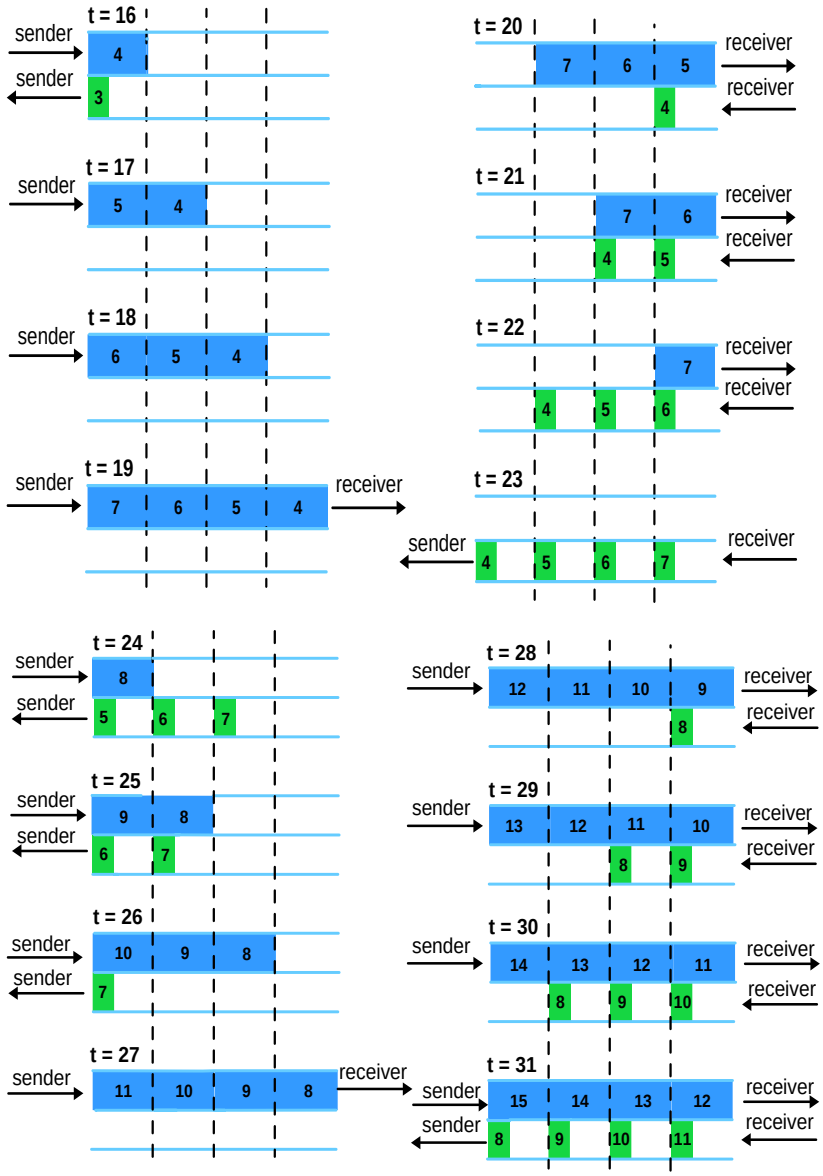
За одређену везу (путању) кашњење због времена простирања је константно, али кашњење због преноса зависи од величине пакета. При малим брзинама преноса доминантно је кашњење због преноса. док је при великим брзинама преноса (нпр. брзине реда Gb/s) доминантан је утицај времена простирања.

Када пошиљалац прими сегмент *Ack* (A_1) он може да пошаље два додатна сегмента (на слици 3.22 означена са 2 и 3). У тренуцима *t*=8 и *t*=9 прозор загушења је два сегмента. Ова два сегмента крећу се удесно ка примаоцу који у тренуцима *t*=12 и *t*=13 шаље *Ack* сегменте. Временски распоред сегмената потврде идентичан је са временским распоредом сегмената података и назива се самотактовање TCP процеса. Пошто пријемник може да генерише потврду (*Ack* сегмент) тек кад пристигну подаци распоред *Ack* сегмената указује на брзину пристизања података на пријемној страни[1].

На слици 3.23 приказано је следећих 16 временских јединица. Када стигну две потврде (*Ack* сегменти) повећава се прозор загушења са два на четири сегмента и четири сегмента (означена као 4, 5, 6 и 7) шаљу се у тренуцима *t*=16 до *t*=19. Прва потврда пристиже у тренутку означеном са *t*=23. Четири *Ack* сегмента увећавају прозор загушења са четири на осам сегмената и сегменати означени као 8, 9, 10, 11, 12, 13, 14, и 15 шаљу се у тренуцима *t*=24 до *t*= 31.

У тренутку *t*=31 и свим следећим тренуцима, пут између пошиљаоца и примаоца је попуњен подацима. Не може да прихвати више података без обзира на прозор загушења или објављени прозор (примаочев). У свакој јединици времена прималац уклања један сегмент са мреже а пошиљалац убацује други сегмент у мрежу. Између пошиљаоца и примаоца у једном правцу се преноси више сегмената података а у супротном правцу једнак број *Ack* сегмената. Ово је идеално стабилно стање везе.

[1] У стварности редови чекања у повратку могу променити брзину пристизања потврде тј. *Ack* сегмената .



Слика 3.23 Тренуци *t*=16 до *t*=31 у преносу веће количине података

Производ пропусног опсега и кашњења

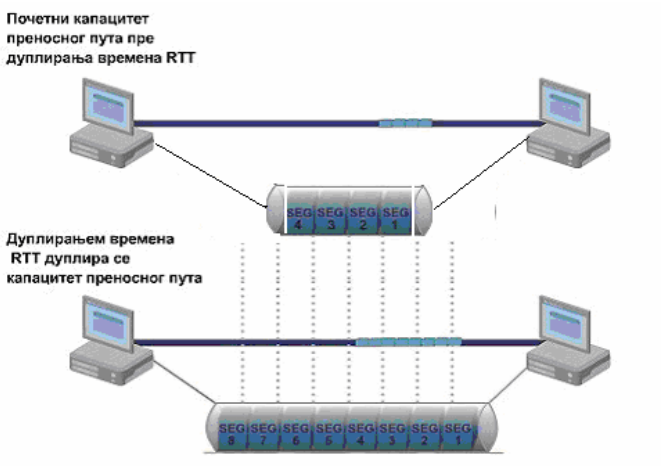
Сада бисмо могли да одговоримо на питање колико велики отвор прозора може да буде? У нашем примеру за максималну пропусну моћ пошиљалац треба да у сваком тренутку

има осам непотврђених сегмената чији је пренос у току. Објављени прозор мора бити исто толико велики пошто толико сегмената пошиљалац може да пошаље. Можемо израчунати капацитет везе (пута) на следећи начин.

Капацитет [у битима] = Пропусни опсег [b/s] ∗ RTT[s] (3.1)

Ова вредност може да се мења у широком опсегу у зависности од брзине трансмисионих медијума у рачунарској мрежи и од укупне вредности кашњења RTT између две посматране тачке мреже. На пример за T1 линије[1] са једног на други крај САД-а и време RTT које се рачуна да је око 60ms добија се капацитет[2] од 11580 бајтова. Ово се сматра прихватљивим[3] са становишта величине меморијског простора. Међутим уколико се за пренос са једног на други крај САД-а користе T3 линије[4] капацитет је 337500 бајтова добијена вредност је већа од максималне могућег прозора примаоца[5] (објављеног прозора). У поглављу 3.3 описана је могућност за проширење величине објављеног прозора користећи опционо поље у заглављу TCP сегмента.

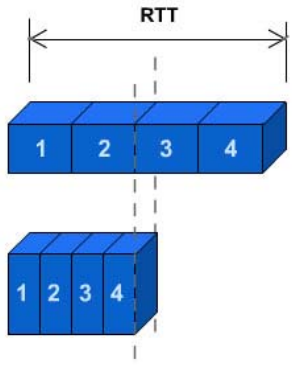
И опсег и кашњење могу да утичу на капацитет преносног пута[6] између пошиљаоца и примаоца. На слици 3.24 приказано је графички како се дуплирањем времена RTT дуплира капацитет.



Слика 3.24 Дуплирањем времена RTT дуплира се капацитет „цеви"

[1] У овој анализи користимо вредност од 1,544 Mb/s. Брзина података је тачније 1,536 Mb/s пошто се први бит (од укупно 193 бита) користи да означи почетак рама.
[2] Среће се и под називом „производ опсега и кашњења" [Stivenson]
[3] Објашњено у поглављу 3.6.14, део „Величина прозора".
[4] У нашој анализи користимо вредност од 45 Mb/s тачније 44,736 Mb/s. Брзина података је 44,210 Mb/s.
[5] 65 535 бајтова
[6] У литератури се пут између пошиљаоца и примаоца назива и цев (*pipe*). Аналогија је погодна за сликовито представљање „паковање" сегмената.

На сличан начин на слици 3.25 приказано је како се двоструким повећањем пропусног опсега дуплира капацитет преносног пута (тзв. цеви).



Слика 3.25 Дуплирањем опсега дуплира се капацитет преносног пута

У доњем делу слике 3.25 претпоставили смо да је брзина рачунарске мреже дуплирана, чиме нам је омогућено да шаљемо четири сегмента за половину времена са горњег дела слике. Поново је капацитет пута удвостручен. Претпоставили смо да сегменти у горњој половини слике имају исти број битова као и сегменти у доњој половини слике.

3.8 Контрола загушености на TCP слоју

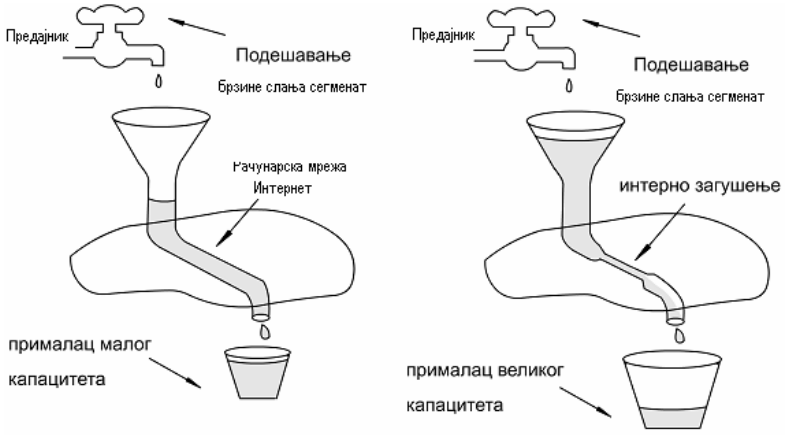
Уколико је оптерећење мреже веће него што она може да поднесе долази до загушености. Ни Интернет не представља изузетак. У овом поглављу анализираћемо алгоритме који се користе за разрешавање овог проблема. Мрежни слој такође покушава да реши проблем загушености, али већину тежих задатака разрешава TCP слој.

Теоретски загушење се може избећи уколико се не шаљу (у рачунарску мрежу тј. Интернет) нови пакети док се одредишту већ претходно послати не испоруче. TCP покушава да реализује овај принцип мењањем величине отвора прозора.

Први корак у решавању проблема загушења је његово проналажење. Кашњење може да буде последица загушења на линији или преоптерећености рутера. Одређивање разлике било би тешко.

Данас је губитак пакета услед грешака у преносу релативно редак пошто су на већини дужих путања постављени оптички каблови. Бежичне рачунарске мреже су посебна тема. Тако је највећи део кашњења на Интернету последица загушења. Сви TCP алгоритми који се користе на Интернету претпостављају да су кашњења изазвана загушењем.

На слици 3.26 приказана је хидраулична илустрација овог проблема[1]. Танка цев води до посуде (примаоца) малог капацитета (слика 3.26а). Док пошиљалац не шаље више воде него што посуда може да прими, вода се неће изливати. На слици 3.26б ограничавајући чинилац није капацитет посуде (примаоца) већ унутрашњи, преносни, капацитет мреже. Ако много воде дође пребрзо излиће се (у овом случају преплављујући левак).



Слика 3.26 Хидрауличка илустрација интерног загушења у мрежи

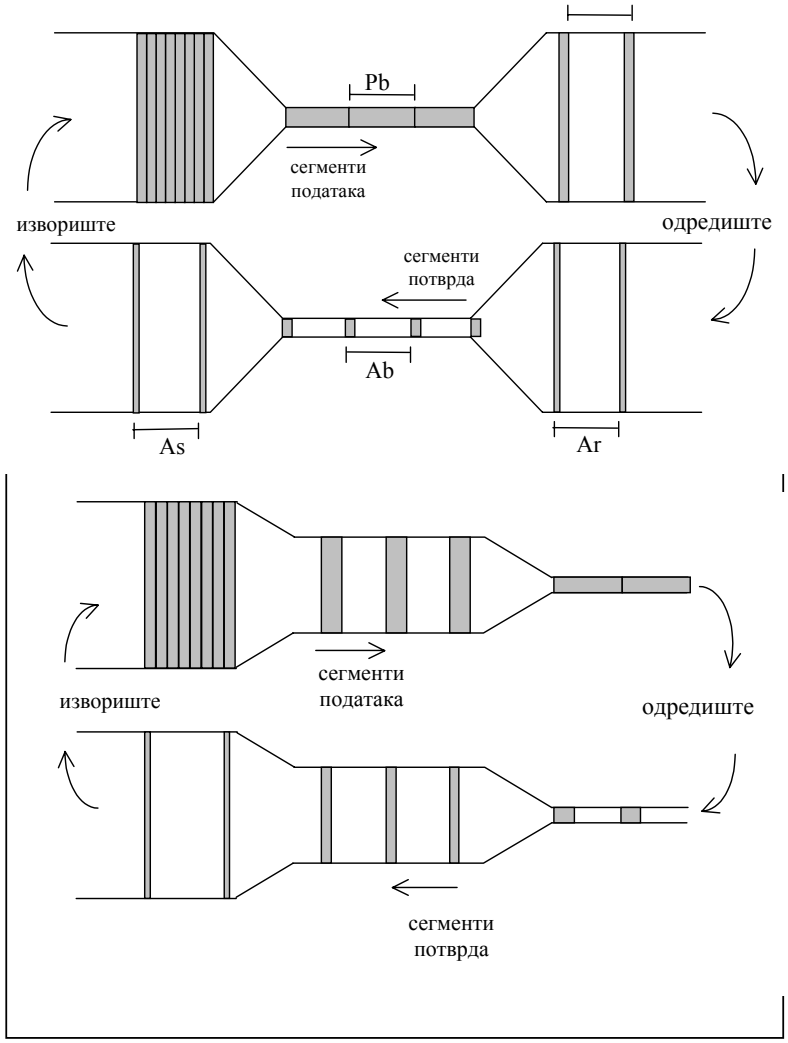
Задатак који Интернет треба да реши јесте како да превазиђе сваки од проблема: капацитет мреже и капацитет примаоца.

Контроле тока

Брзина којом TCP целина шаље податке одређена је брзином пристизања *Ack* сегмената. На брзину испоруке података и *Ack* сегмената могу да утичу тачке загушења (уска грла). На слици[2] 3.27 претпоставља се да је уско грло негде на Интернету. Веза између предајника и пријемника представљена је као цев. Дебљина цеви (висина) пропорционална је брзини којом се преносе подаци кроз њу. Изворište и одредиште налазе се на мрежи великог капацитета и свако од њих може да ради са великим капацитетом. Тањи, централни део цеви, означава или спору линију (линк) негде на Интернету или рутер који је уско грло. Сваки сегмент представљен је правоугаоником чија је површина пропорционална броју битова које садржи. Када сегменти пролазе кроз уску цев они (правоугаоници) се у времену „продужавају"[3]. Време P_b је минимално време (размак) између сегмената на најспоријој вези. Када сегменти стижу на одредиште размак између њих се задржава без обзира што се брзина повећала пошто се време између пристизања сегмената није променило. По томе размак између сегмената на пријему $P_r = P_b$. Уколико одредиште шаље потврду како сегменти пристижу онда је размак *Ack*

[1] [Tanenbaum]
[2] [Jabson]
[3] Време потребно да се пошаље иста количина битова повећава се са смањењем брзине преноса.

сегмената када напуштају пријемник $A_r = P_r$. На крају пошто је размак P_b довољно велики за сегменте података биће довољно велики и за *Ack* сегменте тако да се размак остаје исти на најспоријој вези $A_b = A_r$ и на предајној страни $A_s = A_r$.



Слика 3.27 Размена TCP сегмената када је загушење у саобраћају последица: а) загушења у рачунарској мрежи б) загушења на пријему

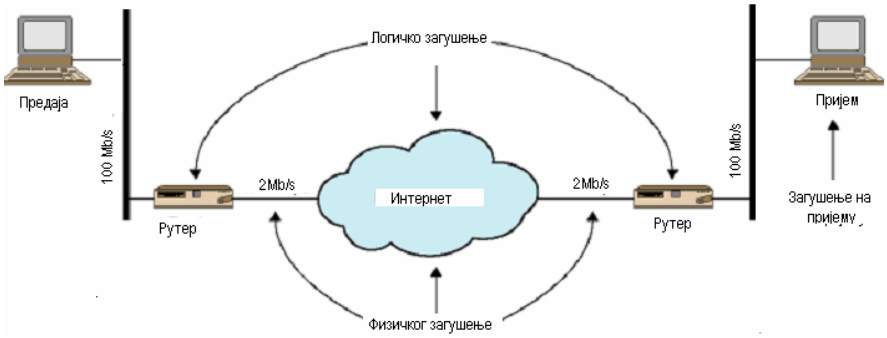
У успостављеном стабилном режиму брзина сегмената података које шаље предајник одговараће брзини пристизања *Ack* сегмената. То значи да је брзина слања сегмената једнака брзини најспорије везе. На овај начин TCP целина аутоматски испитује уско грло

рачунарске мреже и регулише свој проток према томе. Овај процес назива се самотактујуће понашање TCP протокола.

Самотактујуће понашање TCP протокола ради једнако добро ако је уско грло одредиште (пријемник). Претпоставимо да одредиште споро прихвата сегменте било због тога што је заиста споро било што је оптерећено пристизањем сегмената са других веза. На слици 3.27б представљен је овај случај. Претпостављамо да је на најспоријој вези брзина око половине брзине извориште, али је цев на одредишту уска. У овом случају *Ack* сегменти шаљу се брзином којом одредиште прихвата (абсорбује) сегменте података.

Слика 3.27 указује на важан моменат: извориште нема начина да сазна да ли је брзина пристизања сегмената резултат стања на Интернету или резултат стања на одредишту. Прво се превазилази контролом загушења а друго контролом тока.

Уско грло дуж укупног пута између извориште и одредишта (пут сегмента података и пут *Ack* сегмента) јавља се на различитим местима. Дефинише се као логичко или физичко уско грло (слика 3.28). У примеру на слици 3.28 додељује се комплетан капацитет локалне мреже једној TCP вези која има потенцијални пропусни опсег од 100 Mb/s. У овом случају веза E1од 2Mb/s између рутера и Интернета постаје физичко уско грло. Када се успостави стабилно стање расположиви капацитет TCP веза може ефикасно да користи.



Слика 3.28 TCP проток сегмената и контрола загушења

Много чешћа су логичка уска грла и настају као последица редова за чекање у рутерима, рачунарској мрежи, комутаторима или у самом одредишту. Кашњења изазвана чекањем у редовима мењају се у зависности од саобраћаја тј. оптерећења мреже, што доводи до потешкоћа у постизању стабилног стања.

Промене у величини кашњења незаобилазни су део Интернета са IP протоколом. Уколико је TCP проток података мали онда је капацитет Интернета неискоришћен. Уколико једно или више TCP извориште користи превелик капацитет остали TCP токови биће презагушени. Уколико већи број TCP извориште користи превелики капацитет сегменти ће у преносу бити изгубљени што доводи до ретрансмисија. Кашњење потврда због преоптерећења рачунарске мреже такође доводи до беспотребних ретрансмисија. Поновљено слање сегмената повећава саобраћај у мрежи, загушење расте, повећава се

кашњење и повећава број сегмената који се одбацују. Резултат је још више ретрансмисија које стање у мрежи даље погоршавају.

Види се да без обзира што је TCP клизајући прозор пројектован за контролу тока података са једног на други крај рачунарске мреже мора се узети у обзир потреба за контролом (управљањем) загушења. Од када је објављен документ RFC793 бројне технике, које су имале за циљ да побољшају управљање загушењем, биле су имплементиране на TCP слоју. У табели 3.4 дате су најраспрострањеније технике.

Мерење	RFC 1122	TCP Tahoe	TCP Reno
Процена RTT варијансе[1]	√	√	√
Експоненцијално кашњење RTO[2]	√	√	√
Карнов алгоритам[3]	√	√	√
Успорени почетак	√	√	√
Динамичко подешавање прозора приликом загушења[4]	√	√	√
Брза ретрансмисија[5]			√
Брзи опоравак[6]			√

Табела 3.4 Примена TCP мерења загушења

Ниједна од техника из табеле 3.4 не нарушава оригинални TCP стандард. Може се чак констатовати да оне реализују концепте који су у оквиру спецификације за TCP слој. У табели је назначено које од техника су обавезне у RFC1122 а које су примењене у популарним Беркли верзијама UNIX оперативног система.

Технике побројане у табели 3.4 спадају у две категорије: управљање часовницима за ретрансмисију[7] и управљање отвором прозора[8]

[1] RTT *variance estimation*
[2] *Exponential* RTO *backoff*
[3] Karn's algorithm
[4] *Dinamic window sizing*
[5] *Fast retransmit*
[6] *Fast recovery*
[7] *Retransmit timer management*
[8] *Window management*

3. Транспортни слој на Интернету – 3. део

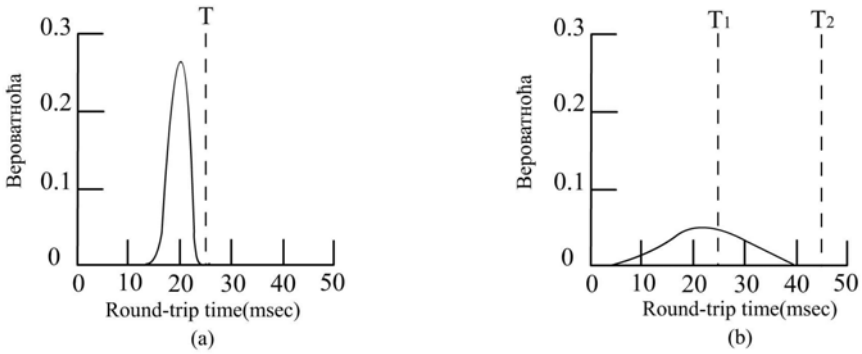
3.9 Управљање часовницима

TCP протокол обезбеђује поуздану услугу транспортног слоај. Један од начина којим се обезбеђује поузданост је коришћење потврде да су подаци стигли на одредиште. Може се десити да се сегменти података и потврда изгубе негде на Интернету. TCP протокол овај проблем разрешава употребом часовника (табела 2.4). Када се сегмент са подацима пошаље часовник се поставља на одређену вредност. Уколико потврда не стигне у том времену (истекне часовник) подаци се поново шаљу (а часовник се поново покреће). Ако је потврда за послати сегмент пристигла пре него што је часовник сигнализирао да је време истекло, часовник се зауставља. Критична тачка примене овог механизма је метод за поновно слање (ретрансмисиона стратегија).

TCP протокол може да управља са четири различита часовника по свакој вези:

- часовник за ретрансмисију користи се када се очекује потврда од супротне стране,
- часовник истрајности води рачуна о размени информација о величини прозора, без обзира да ли је друга страна затворила прозор,
- часовник који води рачуна да ли је одредишна страна у раду,
- часовник 2MSL одређује време колико је веза у стању TIME-WAIT.

Поставља се питање на коју вредност треба часовник поставити? Овај проблем много је сложенији код транспортног слоја на Интернету него код протокола слоја везе. Код слоја везе кашњење је предвидиво (мале су варијације) тако време на које је подешен часовник може имати вредност која је мало већа од оне за које се очекује потврда, као што је то приказано на слици 3.29а. Ако се потврда не појаве у одређеном времену значи да су рам или потврда изгубљени. Потврде на слоју везе ретко касне.



Слика 3.29 Функција расподеле вероватноће пристизања TCP потврда

Функција расподеле вероватноће пристизања TCP потврда приказана је на слици 3.29б. Одређивање ове функције веома је тешко. Ако је време на које је подешен часовник превише кратко (T1) доћи ће до беспотребног поновљеног слања сегмената чиме се саобраћај на Интернету повећањава а самим тим загушења у саобраћају. Ако је време на које је подешен часовник велико (T2) онда, ако је сегмент изгубљен, протекне превише времена пре него што се сегмент поново пошаље. Како се израчунава време на које се часовник подешава и колико често[1] се сегменти поново шаљу?

Решење су динамични алгоритми који стално мењају временски интервал на који се подешава часовник. Промене временског интервала засноване је на константним мерењима на мрежи. Алгоритам који се користи предложио је Јакобсон 1988. године.

У овом поглављу поћи ћемо од једноставног примера времена на који се поставља часовника за поновног слања сегмената (ретрансмисиони часовник), а затим ћемо анализирати типичну примену времена потребног да се пошаље пакет и да стигне потврда за њега (RTT). Анализираћемо и како TCP целина користи измерене вредности да би проценила време на које треба поставити ретрансмисиони часовник, а затим и метод који омогућава да се загушење у мрежи избегне. Ове технике[2] можемо ситематизовати као:

- Мерење времена RTT.
- Карноов алгоритам.
- Експоненцијално RTO кашњење.
- Успорени почетак.
- Алгоритми за избегавање загушења.

[1] У примени „ICMP *port unreachable*" опције и TFTP клијенти који користе UDP протокол користи се једноставан метод поновног слања и подешавање часовника - претпостављ се да је време од 5sec довољно.
[2] Описане су у документу RFC258.

- Брза ретрансмисија.
- Брзи опоравак.

Мерење времена RTT

Основ за постављање времена TCP часовника је мерење укупног времена RTT које протекне од тренутка започињања слања сегмента до пријема потврде за тај сегмент. Очекује се да се вредност RTT мења у времену у зависности од:

- путање коју сегмент пређе и
- промене интензитета саобраћаја у рачунарској мрежи.

TCP целина треба да прати ове промене и према томе мења вредност времена на које се одговарајући часовници постављају. Измерену вредност RTT означићемо са *M*, а њену процену у тренутку посматарања са *R*. Једноставна техника за предвиђање следеће вредности на основу претходних[1] може се представити једначином:

$$R \leftarrow \alpha * R + (1-\alpha) * M \tag{3.2}$$

где је α фактор поравњавањ за који је препоручена вредност од 0,8 до 0,9. Усредњена вредност RTT израчунава се код сваког новог мерења. Значајан део[2] сваке нове процене *R* потиче од претходне процене, а мањи део[3] од мерења *M* које је урађено у том тренутку. Када се добије усредњена процена *R*, која се мења како се мења вредност RTT, препоручује се[4] да се време RTO[5] на које се поставља ретрансмисиони часовник буде дата следећом релацијом:

$$\text{RTO} = \beta * R \tag{3.3}$$

Типична вредност за β=2. У стабилним условима са малом варијансом вредности RTT ова формула доводи до велике вредности RTO. У нестабилним окружењима вредност β=2 није добра заштита од беспотребних ретрансмисија. По Јакобсону беспотребне ретрансмисије су додатно повећање саобраћаја мреже која је већ преоптерећена.

Израчунавање вредности RTO и на основу одступања (варијансе) и на бази средње вредности обезбеђује много бољи одзив на велике промене времена RTT од израчунавања вредности RTO као средње вредности помножене неком константом.

Један од начина је израчунавање стандардне девијације (подразумева израчунавање квадрата и квадратног корена). Једноставнија за израчунавање је средња девијација. Долазимо до следећих једначина које се примењују код сваког мерења времена RTT:

$$E_{rr} = M - A \tag{3.4}$$

---

[1] Специфицирана у документу RFC793.
[2] Од 80% до 90% у зависности од тога колика вредност је узета за α.
[3] Од 20% до 10% у зависности од тога колика вредност је узета за α.
[4] Специфицирана у документу RFC793.
[5] *Retransmission TimeOut value*.

$$A \leftarrow A + g * E_{rr} \quad (3.5)$$

$$D \leftarrow D + h * (|E_{rr}| - D) \quad (3.6)$$

$$RTO = A + 4D \quad (3.7)$$

где је A поравњана вредност RTT (процењена или усредњена) а D поравњана вредност средње девијације. E_{rr} је разлика између измерене вредности управо добијене и текуће процењене вредност RTT. Вредности A и D користе се за израчунавање следећег RTO времена. Параметар g користи се за усредњавање и поставља се на вредност 0,125 (1/8). Параметар h за израчунавање девијације поставља се на вредност 0,25 (1/4). Што је параметар h већи то ће са порастом укупног времена RTT доћи до бржег пораста времена на које се подешава ретрансмисиони часовник[1] RTO.

Упоређујући оригинални метод са Јакобсоновим види се да је израчунавање ублажених вредности слично[2] али је употребљена друга вредност (4 уместо 2). У Јакобсоновом израчунавању вредност RTO зависи од ублажене вредности RTT и ублажене вредности средње девијације, док оригинални метод користи умножак ублажених вредности RTT.

Израчунавање се може обавити једноставним операцијама – сабирањем целих бројева, одузимањем и померањем. Константа 4 је произвољно изабран и има две предности. Множење са 4 код израчунавању вредности RTO рачунар може се обавити једноставном операцијом померања. Друго, избегава се сувишно понављање слања пакета пошто мање од 1% свих послатих пакета стиже у времену дужем од четвороструке стандардне девијације.

Карноов алгоритам

Проблем који настаје приликом динамичке процене RTT је шта урадити када сегмент након истицања времена на часовнику буде поново послат. Када се добије потврда није јасно да ли се она односи на слање првог или поновљеног сегмента. Погрешан закључак могао би да озбиљно угрози процену вредности RTT. Фил Карн, радио - аматер, ентузијаста, био је заинтересован за слање TCP/IP пакета радио - везом. Код преноса непоузданим медијумом, радио - везом, на одредиште у најбољем случају стигне 50% пакета. Карн је предложио решење: алгоритам који је по њему добио име и који се користи у већини TCP/IP реализација. Његов предлог био је једноставан:

- не ажурирати вредност RTT по поновљеном слању сегмента,
- време на које се поставља часовник за поновно слање (RTO) удвостручити при сваком неуспелом слању (као у једначини 3.3), све док се не добије потврда за сегменте који стигну на одредиште из првог покушаја.

Експоненцијално RTO кашњење

[1] Јакобсон је у својим првим истраживањима 1988. год. предложио за вредност константе у једначини (3.7) 2 а касније 1990. год. вредност 4. У већини примена користи се вредност 4.

[2] $\alpha = 1 - g$

Када се сегмент пошаље а потврда за њега не стигне у времену RTO, на које је постављен часовник за ретрансмисију, TCP целина мора поново да пошаље сегмент. У документу RFC793 претдложено је да се при поновном слању сегмента часовник постави на исто време RTO. Дугачко време у коме потврда није стигла последица је загушења у мрежи па се постављање часовника на исту вредност не препоручује.

Замислимо следећи сценарио: постоји велики број активних TCP веза из различитих извора и све заједно доприносе саобраћају на Интернету. Ефекат загушења има утицаја на сваку од ових веза. То значи да ће готово истовремено много сегмената бити поново послато у мрежу што ће продужити или чак погоршати загушење у саобраћају.

Метод који се предлаже (различит од метода у документу RFC793) је да се вредност RTO повећава при свакој ретрансмисији. На тај начин даје се времена да се загушење на Интеренту регулише. Једноставан метод који је примењен је:

$$RTO \leftarrow k*RTO \tag{3.8}$$

Ова релација доводи до експоненцијалног раста времена RTO са сваком следећом ретрансмисијом. Уобичајена вредност за параметар k је 2. Са овом вредношћу параметра k метод се назива бинарно експоненцијални. Исти метод је примењен и код CSMA/CD протокола у локалним рачунарским мрежама.

Алгоритам за избегавање загушења

Успорени почетак, који је описан у једном од претходних поглавља, начин је за иницирање протока података кроз TCP везу. У некој тачки дођи ће се до граничног капацитета неког од рутера и он ће једноставно одбацити сегмент. Избегавање загушења је начин који се примењује када дође до губитка сегмената. Описао га је Јакобсон.

Претпоставка Јакобсоновог алгоритма је да је губитак сегмената због грешке (оштећења) на њима веома мали (много мањи од 1%). Значи да губитак пакета указује на загушење негде у мрежи између изворишта и одредишта. Постоје два показатеља да је дошло до губитка сегмента: истицање часовника и пријем дупликата *Ack* сегмента.

Избегавање загушења и успорени почетак независни су алгоритми са различитим циљевима. Али када дође до загушења у саобраћају треба смањити брзину слања података а затим применити алгоритам успорени почетак. У пракси ова два алгоритма користе се заједно.

Избегавање загушења и успорени почетак захтевају да се прате две променљиве за сваку везу: прозор загушења (*cwnd*[1]) и преломна вредност успореног почетка (*sstresh*[2]). Комбинација ова два алгоритма ради на следећи начин:

- У току иницијализације везе поставити *cwnd* на величину једног сегмента а вредност *sstresh* на 65535 бајтова (62k);

[1] *Congestion window*
[2] *Slow start treshold size*

- Предајна TCP целина никада не шаље више сегмената од минимума величине прозора загушења и величине објављеног прозора (тј. прозора пријемне TCP целине). Избегавањем загушења реализује се контрола тока на страни предајника а објављеним прозором реализује се контрола тока на страни пријемника. Прва је заснована на процени предајника о загушењу саобраћаја у мрежи; друга је везана за расположив меморијски простор на страни пријемника (погледати слику 3.6);
- Када дође до загушења (на које указује истицање часовника или дупликат *Ack* сегмента) половина текуће величине прозора (минимум од *cwnd* и пријемниковог објављеног прозора, али најмање два сегмента) се памти као *sstresh*. Уколико на загушење указује истицање часовника *cwnd* се поставља на један сегмент (тј. успорени почетак);
- Када су нови подаци потврђени од друге стране увећава се *cwnd*, али начин на који се увећава зависи од тога да ли је примењен алгоритам успореног почетка или избегавање загушења.

Уколико је *cwnd* мањи или једнак као *sstresh* ради се са успореним почетком; у супротном ради се са избегавањем загушења. Успорени почетак примењује се све док се не стигне до тачке која је на половини оне вредности која је била када је дошло до загушења (записана је вредност половине отвора прозора која је довела до проблема у кораку 2). Даље се наставља са алгоритмом за избегавање загушења.

Код успореног почетка *cwnd* започиње са једним сегментом и увећава се за по један сегмент сваки пут када стигне *Ack* сегмент. На тај начин се отвор прозора увећава експоненцијално: шаље се један сегмент, па два, па четири итд.

Избегавање загушења диктира да се вредност *cwnd* увећава за 1 када стигне *Ack* сегмент али тек по истеку времена једнаког једном RTT. Ово је адитивно увећавање за разлику од успореног почетка које је експоненцијално увећавање. Ми желимо да увећамо *cwnd* највише за један сегмент после укупног истека времена RTT независно од тога колико је потврда примљено. Избегавање загушења се наставља све док се не детектује загушење. Формула која се користи за израчунавање[1] прозора загушења може се представити на следећи начин:

$$cwnd \leftarrow MSS*MSS/cwnd \qquad (3.9)$$

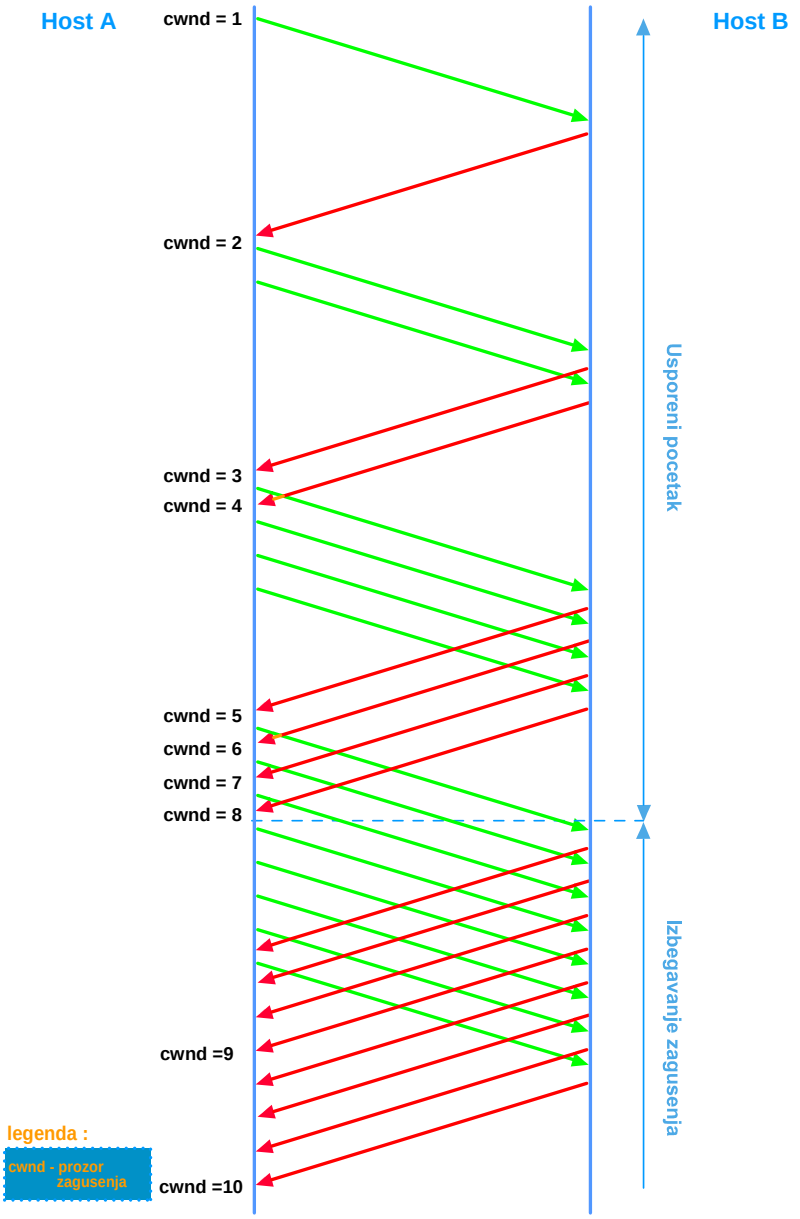
Ово прилагођавање се извршава сваки пут када пристигне *Ack* сегмент (који није дупликат). Једначина обезбеђује прихватљиву апроксимацију принципа о увећавању прозора загушења (*cwnd*) за величину једног сегмента података по времену RTT. На пример уколико је MSS величине 1460 бајтова, и прозор загушења 14600 бајтова онда се 10 сегмената података може послати за време једног RTT. Сваки примљени *Ack* сегмент[2] повећава прозор загушења за 1/10 MSS. После 10 примљених сегмената вредност прозора загушења се повећава за 1 као што је и тражено.

[1] Детаљније се може наћи у документу RFC2581.

[2] Претпостављајући да за сваки примљени сегмент података пријемник шаље један *Ack* сегмент.

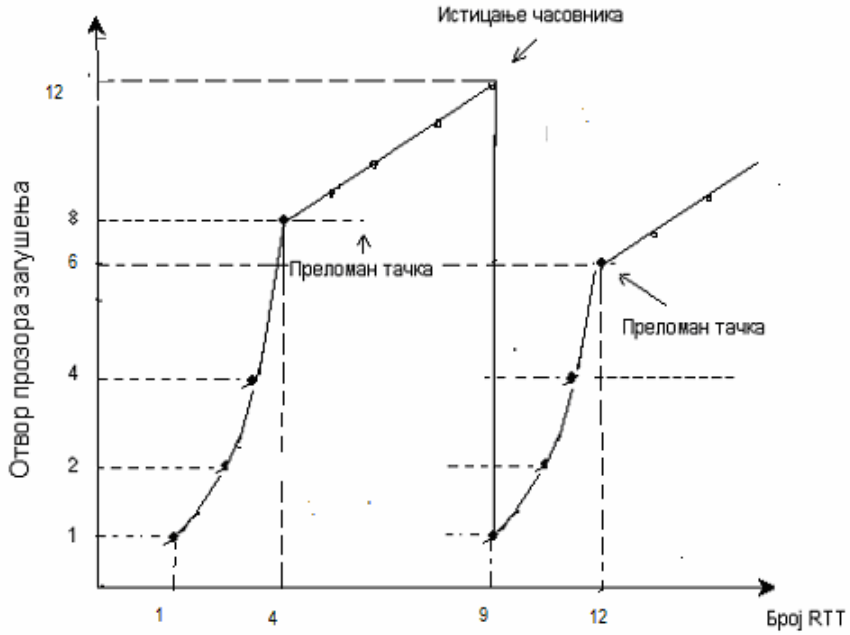
Успорени почетак ће увећати *cwnd* за онолико колико је потврда (*Ack* сегмената) примљено у RTT времену.

Слика 3.30 приказује коришћење успореног почетка. Лева страна слике понавља з 3.21. Као додатак овој слици је сегмент на крају који ће довести до истицања времена на који је постављен часовник. Понашање после истицања времена представљено је на слици 3.30б. Вредност *sstresh* је постављена на вредност 8. Док се ова граница не достигне TCP користи алгоритам успореног (експоненцијалног) почетка да би проширио прозор загушења. Када се достигне *sstresh* вредност *cwnd* се увећава линеарно.



Слика 3.30 Примена алгоритама успорени почетак и избегавање загушења

Као илустрацију рада описаних алгоритама загушења погледајмо и слику 3.31. Максимална величина сегмента износи 1024 бајта. Иницијално се отвор прозора загушења поставља на 64k. У тренуткку посматрања (слика 3.31) *sstresh* је постављен на 8k, отвор прозора загушења на 1k за слање редног броја 0. Затим величина прозора загушења расте експоненцијално до вредности *sstresh* 8k. Од те тачке наставља са линеаран раст.



Слика 3.31 Илустрација рада алгоритма загушења

Код 8. слања време на које је часовник постављен истиче. Вредност *sstresh* се поставља на половину тренутне величине прозора загушења (која је сада 12k, па је *sstresh* 6k). И поново се прелази на успорени почетак. Када стигне потврда за 12. слање (нови *sstresh)*, прва четири слања су већ удвостручиле прозор, после чега његов раст постаје опет линеаран.

Величина прозора загушења ће расти до величине прозора примаоца уколико у међувремену не истекне време на које је часовник подешен. Када достигне ту величину престаје са растом и остаје константна све док се не промени величина прозора примаоца[1] или часовник не сигнализира истицање времена на који је подешен.

[1] Ако TCP целина добије ICMP SOURCE QUENCH пакет третира се као сигнализирање часовника да је време истекло.

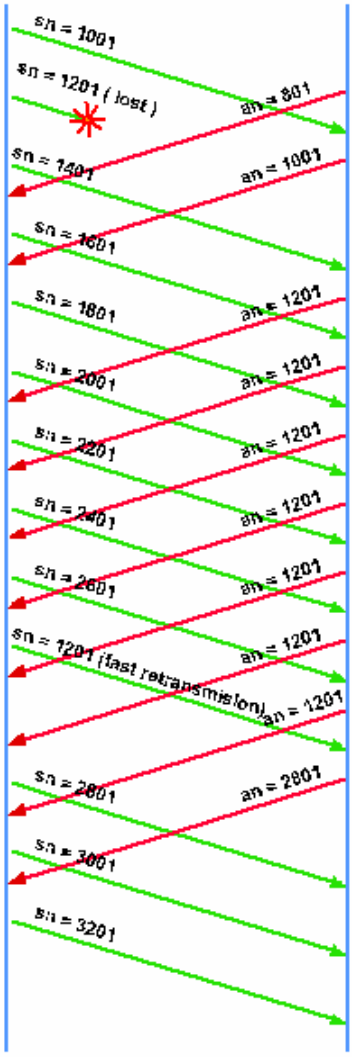
Брзо поновно слање

Са брзим поновним слањем сегмената са подацима избегава се да TCP целина чека да истекне време на које је подешен часовник за ретрансмисију па да понови слање сегмента. Измену алгоритма за избегавање загушења предложио је Јакобсон 1990. год.

Пре него што објасниму у чему се састоје измене подсетимо се да TCP целина може да пошаље потврде (дупликат *Ack* сегмента) када добије сегмент са подацима који са редним бројем који не очекује (ван редоследа). Пријемна TCP целина шаље *Ack* дупликат чим добије први сегмент података је ван редоследа). Његова намена је да укаже пошиљаоцу (другом крају везе) да је примљен сегмент ван редоследа и да му укаже који редни број он очекује.

Предајна TCP целина не зна да ли је дупликат *Ack* сегмента последица изгубљеног сегмента података или последица само пријема сегмената података у неправилногм редоследу. Ако су сегменти података примљени у неправилном редоследу пријемник шаље дупликате *Ack* сегмента све док не обради примљене сегменте података. Тада шаље *Ack* сегмент којим потврђује све примљене сегменте података.

Уколико предајна TCP целина добије три дупликата *Ack* сегмента један за дугим онда она претпоставља да је сегмент податак са редним бројем као у *Ack* сегменту изгубљен. Поново шаље тај сегмент и не чека да часовник за ретрансмисију сигнализира да је време истекло. Овај метод се назива брзи алгоритам за поновно слање (ретрансмисију).



Слика 3.32 Брзо поновно слање

Слика 3.32 приказује процес брзе ретрансмисије. Станица C_1 шаље сегменате са подацима од по 200 бајтова. Сегмент података са редним бројем 1201 је изгубљен, али станица C_1 неће да реагује док време на које је подешен часовник RTO не истекне, или док станица C_2 не затвори свој прозор. Станица C_2 прима сегменте 1001 (редни бројеви од SN=1001 до SN=1200) и потврђује их са *Ack* сегментом (редни број AN=1201). Затим прима сегмент 1401 (редни бројеви од SN=1401 до SN=1600). Пошто је овај сегмент изван редоследа станица C_2 понавља *Ack* сегмент (AN=1201) и понављаће га све док се

не појави сегмент са реднимним бројем 1201. За то време станица C_1 је примила четири дупликата *Ack* сегмента (AN=1201) и послала седам сегмената података са редним бројевима иза сегмента 1201. Станица C_1 поново шаље сегмент података 1201 (редни бројеви од SN=1201 до SN=1400) и наставља са слањем.

Брзи опоравак

Када TCP целина поново пошаље сегмент података користећи брзу ретрансмисију, она зна (тј. претпоставља) да је сегмент изгубљен без обзира што није протекло време на које је подешен часовник за ретрансмисију за тај сегмент. Један од начина би био да се од тог тренутка надаље примене алгоритми успореног почетка и заобилажења загушења (слике 3.30 и 3.31). Овај приступ је превише конзервативан. пошто *Ack* дупликати указују TCP целини да постоји проток података између два краја везе. Пријемник може да генерише *Ack* дупликат тек када неки од по реодследу следећи сегмент података стигне и буде смештен у меморијски простор пријемника. TCP целина не жели да употребом успореног почетка беспотребно нагло смањи проток. Због тога је Јакобсон предложио модификацију: поново послати изгубљени сегмент, прозор загушења (*cwnd*) поставити на половину текуће вредности, и наставити са линераним увећањем прозора загушења. На тај начин се алгоритми брзе ретрансмисије и брзог опоравка се примењују заједно што се прецизно може описати на следећи начин:

1. Када се један за другим приме три *Ack* дупликат, поставити *sstresh* на половину минимума текућег прозора загушења (*cwnd*) и пријемног објављеног прозора али не мањег од два сегмента.

   Поново послати недостајући сегмент.

   Поставити *cwnd* на *sstresh* и додати троструку вредност величине сегмента. Овим се увећава прозор загушења за онолико сегмената колико је напустило мрежу и пристигло на одредиште.

2. Сваки пут када пристигне нови дупликат *Ack* сегмента, увећати *cwnd* за величину сегмента. Овим се повећава прозор загушења за додатну величину сегмента који је напустио мрежу. Послати пакет, уколико је дозвољено, са новом вредношћу *cwnd*.

3. Када стигне следећи *Ack* сегмента, потврда за нове податке, поставити *cwnd* на *sstresh* (вредност постављену у 1. кораку). Ово ће бити потврда за поновно слање из 1. корака, једно укупно време RTT после ретрансмисије. Поред тога, овај *Ack* сегмент требало би да потврђује све сегменте који су послати између последњег изгубљеног пакета и пријема трећег *Ack* дупликата.

   Ови кораци су заобилажење загушења пошто се успорава слање на половину брзине слања у тренутку када је пакет изгубљен.

Правци даљег развоја алгоритама за избегавање загушења

Важно је нагласити да постоји потреба да се алгоритми за избегавање загушења мењају и усавршавају и прилагоде TCP везама великих брзина. Средња вредност величине

прозора загушења код TCP веза у стационарном стању са вероватноћом изгубљених пакета *p* може се изразити формулом:

$cwnd \approx 1{,}2 / p^{1/2}$

Пратећи анализу описану у документу RFC3649 види се да постоје озбиљна ограничења у величини прозора загушења који се може достићи у реалним условима. На пример посматрајмо TCP везу:

- у којој се размењују сегменти величине 1500 бајтова,
- у којој је кашњење RTT 100ms и
- проток од 10Gb/s.

Да би се достигло стационарно стање у оваквој TCP вези потребно је да је средња вредност прозора загушења 83333 сегмента. Такође се захтева и да не дође до одбацивања више од једног пакета на сваких $5*10^9$ пакета што је еквивалентно вероватноћи да дође до губљења пакета од $2*10^{-10}$ или вероватноћи битске грешке од $2*10^{-14}$. Ово су нереални захтеви за рачунарске мреже које су данас у употреби. Због тога је покретнут већи број истраживања која су довела до нових верзија TCP протокола која су пројектована за рачунарске мреже великих брзина.

Коректност[1]

Посматрајмо *k* TCP веза таквих да свака од њих има различито кашњење с краја на крај и да свака од њих пролази кроз исти линк – уско грло који је брзине R b/s. Претпоставимо да се сваком од веза преносе велике датотеке и да се истим линком преносе подаци који користе UDP протокол. Механизам за контролу загушења се сматра коректним уколико је срења брзина преноса сваке од веза R/*k*. То значи свака од веза добија исти део пропусног опсега линка.

Часовник истрајности[2]

Видели смо да предајна TCP целина шаље података у зависности од количине коју је пријемна целина у стању да прими односно од величине прозора. Шта се дешава када се пријемник огласи да не може да прима податке тј. огласи прозор величине 0? Зауставља се слање предајника док пријемник не огласи прозор различит од 0. Овакав сценарио смо видели на слици 3.6. После пријема сегмента са отвором прозора $W \neq 0$ заустављена наставља се са слањем података. TCP целина мора да предвиди активности за случај да се може десити да се сегмент са $W \neq 0$ изгуби. Шта се може десити:

- да је предајник заустављен и да чека сегмент са отвором прозора $W \neq 0$;
- да је пријемик послао сегмент са отвором прозора $W \neq 0$ и да чека да пристигну нови подаци од предајника.

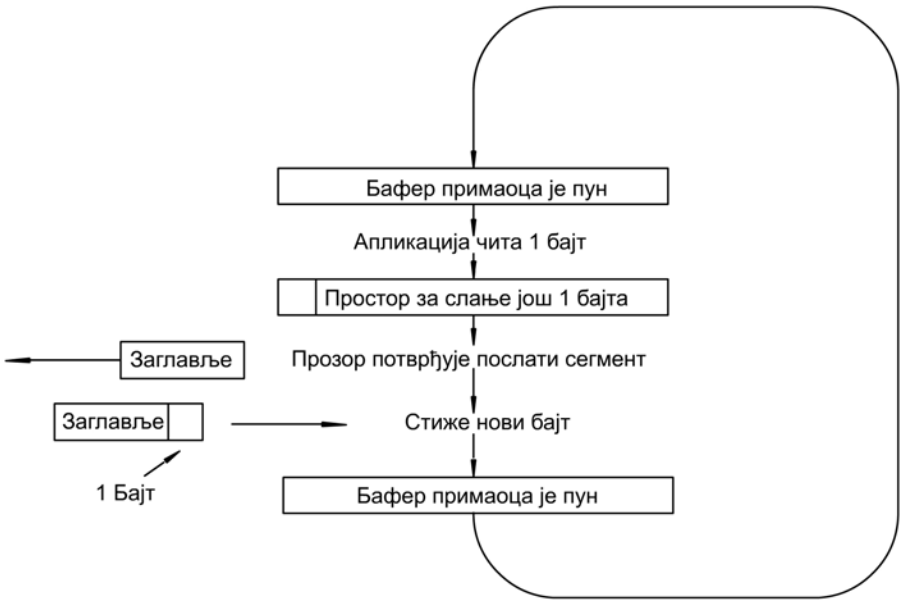
---

[1] *Fairness*
[2] *Persist timer*

Да би се онемогућила да дође до овакве затворене петље користи се часовник истрајности. Предајна TCP целина периодично шаље упит пријемнику да би сазнала да ли је пријемник повећао свој отвор прозора (W≠0). Сегменти који се користе као упит називају се „испитивање прозора[1]". Карактеристика часовника истрајности је да он никада не одустаје од слања упита. Он наставља са слањем упита сваких 60sec све док предајник не отвори прозор или апликација, која везу користи је непрекине.

Ефекат малог прозора[2]

Још један проблем који може настати на TCP слоју као последица употребе отвора прозора за контролу тока је ефекат малог прозора (SWS[3]). Овај проблем настаје када подаци стижу до TCP целине пошиљаоца у великим блоковима, али интерактивна апликација на пријемној страни чита податке бајт по бајту. Да би анализирали проблем погледајмо слику 3.33 У почетку, меморијски простор TCP целине на страни примаоца је пун и пошиљалац је упознат са тим (нпр. величина отвора прозора му је једнака нули). У том случају, интерактивна апликација чита један карактер из TCP низа. Ова акција проузрокује да TCP примаоца мења отвор прозора (један бајт) и обавештава пошиљаоца да може да пошаље један бајт. Пошиљалац прима поруку и шаље један бајт. Меморисјки простор је сада поново пун, пријемна страна шаље потврду, али поставља величину отвора прозора на нулу. Ово може да траје у недоглед.



[1] *Window probes*
[2] *Silly window syndrome* - SWS
[3] Описао је Кларк (Clark)1982.

Слика.3.33 Синдром малог прозора[1]

Кларк је је предложио решење које се састоји у томе да се спречи пошиљалац да шаље промену отвора прозора за само један бајт. Уместо тога он чека да се значајни меморијски простор ослободи и тек тада обавештава пошиљаоца.

Алгоритам Нигла и решење Кларка су комплементарна. Нигл је покушао да реши проблем који је изазвала апликацаија пошиљаоца шаљући TCP целин бајт по бајт. Кларк је покушао да реши проблем апликације примаоца која од TCP целине очитава податке бајт по бајт. Оба решења могу се користити истовремено. Суштина је да пошиљалац не треба да шаље мале сегменте а прималац не треба да пружима мале сегменте.

Часовник активности[2]

Може да се деси да ни један од процеса на крајевима TCP везе не шаље податке и две TCP целине не размењују ништа. То значи да може да се деси да неко покрене процес на страни клијента који успоставља TCP везу са сервером и не настави са радом сатима, данима и месецима а TCP веза ће остати успостављена. Рутери и линкови (нпр. телефонске везе) на путу између два краја везе могу међувремену да прекину и наставе са радом али ова веза се сматра успостављеном све док је станице на крајевима везе не прекину.

Значи претпоставка је да ниједна од апликација (ни клијент ни сервер) нема часовник на апликационом слоју који може да уочи да активност на вези не постоји и да везу треба прекинути. У следећем анализи која следи видећемо да код спољашњих протокола за рутирање (BGP[3]) апликације шаљу тест сваких 30 sec. Часовник који иницира слање овог теста везан је само за BGP протокол и независтан је од рада часовника активности TCP слоја.

Постоје ситуације када сервер жели да зна да ли је станица - клијент у квару и завршила је са радом или је у квару али ће наставити са радом. Часовник активности, који је укључен у многе примене, то омогућава. Часовник активности није део TCP спецификације. Разлог је што се сматра да:

- могу да изазову прекид везе која је неактивна због тренутних али решивих проблема међувеза,
- користе без потребе капацитет везе и додатна су инвестиција.

Постоје разлози за и против коришћења часовника активности. пример који указује да је потребан часовника активности је када је клијента-персоналног рачунар повезан са сервером Telnet протоколом. Ако на крају дана корисник искључи рачунар а да се претходно не одјавио, он оставља полуотворену везу. Сервер може у недоглед да чека

[1] *Silly window syndrome*
[2] *Keepalive timer*
[3] *Borderr Gateway Protocol*

податке од клијента. Употреба часовника активности на страни сервера омогућиће откривање овакве полуотворене везе.

3.10 Карактеристике и правци развоја TCP протокола

TCP протокол дуго је користи везе малих брзина: од 1200b/s на комутираним телефонским линијама до у 100Mb/s локалним рачунарским мрежама. Иако TCP ради и на већим брзинама као што T3, E3, FDDI и мреже гигабитских брзина морају се узети у обзир и одређена ограничења. У овом одељку анализираћемо неке од предложених измена у TCP протоколу које омогућавају максималну пропусну моћ када се ради са великим брзинама.

Откривање максималне јединице за пренос

Comment [v1]: довде сам стигла 19.25 17.8.2007

Када су две станице у истој мрежидефинисана је максимална јединица података која се може том мрежом преносити (MTU[1]). Када се станице комуницирају преко разли читих мрежа сваки линк може да има различиту максимална јединицу преноса. Важна величина је максимална јединицу података која се може пренети целокупном путањом између те две станице (не максимална јединица преноса у мрежи у којој се станице налазе). Ова величина се назива се максимална јединица преноса пута[2] и није константна. Зависи од путање којом се пренос података одвија. Рутирање не мора да буде симетрично па се максимална јединица пута може разликовати у различитим правцима исте везе.

У документу RFC1191[3] је дефинисан мехнаизам за откривање путње[4] јединице као начин за одређивање јединице MTU у било ком тренутку.

Откривање MTU пута на TCP слоју ради на следћи начин: када се веза успостави TCP целина користи MTU вредност излазног интерфејса или MSS који је други крај објавио као почетну величину сегмента. Одређивање MTU пута не дозвољава TCP целини да прекорачи вредност коју је други крај објавио. Уколико други крај није специфицирао MSS користи се подразумевана вредност 536.

Једном када је величина сегмента одабрана сви IP пакети (датаграми) који се шаљу том TCP везом имају постављен DF бит на вредност 1. Уколико неки од рутера на путу мора да дели IP пакета који има DF=1 он га одбацује и шаље ICMP поруку[5] изворишту „не могу да делим пакет".

Пошто се путање (руте) могу динамички мењати, када протекне одређено време може се покушати са повећањем величине MTU (само до вредности MTU коју је одредидиште објавило). У документу RFC1191 препоручено је да овај временски интервал буде 10 минута.

Дугачке везе великог капацитета

---

[1] *Maximum Transfer Unit* - максимална јединица преноса
[2] *Path* MTU
[3] [Moguel and Deering 1990]
[4] *Path* MTU *discovery mechanism*
[5] Детаљније о ICMP протоколу биће речи у следећем поглављу.

Капацитет везе (линка) назива се „производ опсега и кашњења" и дефинише се као:

$Капацитет_{бит}$ = пропусни $опцег_{b/s}$ * RTT_{sec}

Користи се и термин „величина цеви" између две крајње тачке. За различите мреже у табели 3.5. приказан је производ пропусног опсега и кашњења у бајтовима пошто је то начин како се уобичајено мери величина меморијског простора (бафера) и величине отвор прозора на сваком од крајева.

Мрежа	Опсег [b/s]	RTT[1] [b/s]	Производ опсега и кашњења [бајт]
Локална рачунарска мрежа IEEE802.3	10000000	3	3750
T1 телефонске линије, међуконтиненталне	1544000	60	11580
T1 телефонске линије, сателитске	1544000	500	96500
T3 телефонске линије, сателитске	45000000	60	337500
Гигабит, међуконтиненталан	1 00000 0000		

Табела 3.5 Различите мреже и одговарајући производ опсега и кашњења

Мреже са великим производом пропусног опсега и кашњења називају се мреже велике ширине (LFN[2]). TCP веза која користи мреже велике ширине назива се велика широка цев[3]. Ако погледамо слике 3.24 и 3.25 цев се може проширити у хоризонталном правцу (веће RTT) или се може проширити у вертикалном правцу (шири пропусни опсег) или у оба правца. Бројни проблеми се јављају код дугачких широких цеви. Поменућемо неке од њих:

Величина TCP отвора прозора одређена је 16-битним пољем у TCP заглављу што значи да је максимална величина TCP отвора прозора 2^{16} = 65535 бајтова. Као што можемо видети из последње колоне табеле 3.6.5 постојеће мреже већ захтевају већи прозор да би обезбедиле максималну пропусну моћ[4].

Губици пакета у мрежама велике ширине могу значајно да умање пропусну моћ мреже. Уколико је само један сегмент изгубљен алгоритми „брза ретрансмисија" и „брзи

[1] *Round Trip Time*
[2] *Long Fat Networks* - велике дебеле мреже а користи се и енг. термин elefan(t)s.
[3] *Long fat pipe* - велика дебела цев.
[4] Видели смо да је у пољу опције заглавља TCP сегмента предвиђена могућност за повећање максималне величине прозора.

опоравак", који су описани у претходном поглављу, потребни су да би се у вези (цеви) задржали проток пакета. Чак и са овим алгоритмима губитак више од једног пакета у оквиру једног прозора обично доводи до заустављања протока у вези (цеви). Селективна потврда (SACK) предложена је у документима RFC1072[1] да би разрешила проблем губитка више пакета у оквиру једног прозора. Ова опција је изостављена у RFC1323 због техничких проблема.

Видели смо да многе примене TCP протокола мере једно RTT време по прозору а не за сваки сегмент у оквиру прозора. За мреже велике ширине потребна су чешћа мерења RTT времена.

Као што је већ објашњено TCP целина означава сваки бајт података са 32 битним редним бројем. Шта спречава сегменте који су у кашњењу, и чија је веза окончана, да се појаве у новоуспостављеној вези између две станице и два порта као из претходно окончане везе? Прво подсетимо се да TTL поље у заглављу IP пакета поставља горњу границу времена његовог постојања у мрежи - 255 прелазака преко рутера или 255sec. У одељку 3.5 дефинисали смо максимално време трајања сегмента (MSL) као параметар који спречава да до одбацивања IP пакета дође.

У мрежама велике ширине јавља се и проблем са редним бројевима TCP сегмената. Пошто је простор редних бројева коначан исти секвенцијски број се поново користи после пошто се пошаље 4294967296 бајтова. Шта ће се десити ако сегмент једне TCP везе који садржи бајт са редним бројем *N* има велико кашњење и стигне на одредиште у току трајање те везе? До проблема може да дође само уколико се редни број *N* поново користи (нови круг) пре истека периода од MLS. У Етернет локалним рачунарским мрежама потребно је скоро 60 минута да би се послало толико података тако да не може да дође до поновног коришћења истих редних бројева. Време које протекне до поновног корошћења истих редних бројева се смањује како се повећава пропусни опсег преносних система. На пример за T3 преносне системе, брзине 45Mb/s до новог круга коришћења редних бројева долази после 12 минута, а код локалних рачунарских мрежа брзине 1Gb/s после 34 секунде. Проблем код оваквих мрежа није производ пропусног опсега и кашњења већ сам пропусни опсег.

Један од начина да се заштити од овог проблема је су алгоритми који користе опције са временским ознакама сегмената

Наставак

Comment [N2]: Курсор 275

[1] [Jacobson and Braden]

4. МРЕЖНИ СЛОЈ

Мрежни слој је задужен за пребацивање пакета и одабир оптималних путеве од извора до одредишта. Према томе, мрежни слој је најнижи слој који се бави преносом са једног краја мреже на други крај. Да би постигао додељени му задатак мрежни слој треба да буде упознат са:

- топологијом подмреже,
- распоредом и везама свих уређаја који повезују подмреже.

У следећим поглављима анализираћемо неке од карактеристика мрежног слоја које су битне за пројектовање рачунарских мрежа и сервисе које треба да обезбеди транспортном слоју

Комутација пакета

Систем у коме ради протокол мрежног слоја представљен је на слици 4.1. Главне компонента система који називамо подмрежа су:

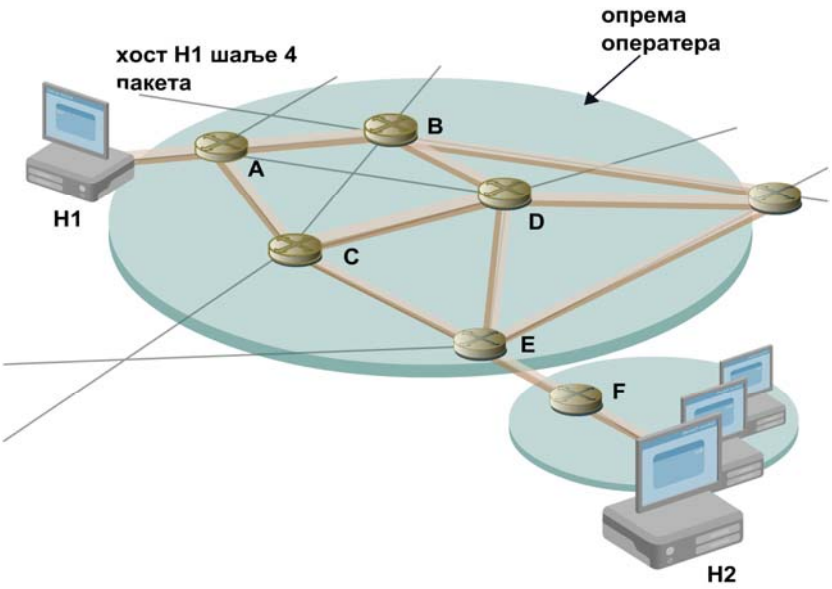
- чворови тј. комутациони уређаји - рутери,
- трансмисионим линије које повезују чворове приказане унутар осенчене елипсе.

Крајњи уређаји који желе да међусобно комуницирају називају се крајње станице. Станица H_1 је директно везан са рутером A подмреже преко изнајмљене линије. Са друге стране станица H_2 се налази на локалној рачунарској мрежи корисника која је са рутером F повезан на подмрежу. Овај рутер такође има изнајмљену линију до подмреже. Описани систем: подмрежа, станице, локална рачунарска мрежа и чворови (рутери) користе се на следећи начин:

- крајња станица (хост) која има пакет за слање шаље га најближем чвору (рутеру), било на својој мрежи или преко тачка-тачка везе до подмреже,
- чвор (рутер) прихвата пакет, смешта га у локалну меморију и проверава да ли је са грешком,
- уколико је пакет без грешке шаље се следећем чвору све док не дође до одредишне станице.

Описани процес назива се комутација пакета „смести и проследи”[1].

[1] *Store and Forward*



Слика 4.1. Окружење протокола мрежног слоја

Сервиси који се пружа транспортном слоју

Мрежни слој пружа сервисе транспортном слоју на интерфејсу мрежни слој/транспортни слој. Сервис мрежног слоја је пројектовани да задовољи следеће циљеве:

- сервиси треба да је независни од начина на који се пакети преусмеравају,
- транспортни слој треба да је заштићен од броја, типа и топологије чворова мреже,
- мрежне адресе доступне транспортном слоју треба да користе јединствени (униформни) бројни систем без обзира да ли се остварује преко локалних рачунарских мрежа или мрежа ширег градског подручја.

С овим циљевима на уму, пројектанти мрежног слоја имају доста слободе у спецификацији сервиса који се нуде транспортном слоју. Основни проблем се своди на да ли мрежни слој треба да нуди сервис без успоставе везе[1] или сервис са успоставом везе[2].

Интернет заједници и пројектанти на њу ослоњени тврде да је једини задатак чворова (рутери) да преусмере (комутирају) пакете. Из њиховог тридесетгодишњег

[1] *Connectionless*
[2] *Connection-oriented*

искуства подмрежа је непоуздана без обзира како је пројектована. Зато крајње станице треба да имају то на уму и раде проверу (детекцију и корекцију) грешака и контролу тока.

Ово гледиште наводи на закључак да би мрежни сервис требало да буде без успоставе везе. Водити рачуна о контроли тока и редоследу пакета је сувишно пошто то већ и станице раде. Претпоставка је да сваки пакет мора да носи пуну адресу одредишта јер се шаље независно од својих претходника.

Друга групација (коју чине телефонске компаније) залаже се да подмрежа треба да обезбеди поуздан сервис са успоставом везе. Тврде да је 100 година успешног искуства са светском телефонском мрежом за такво опредељење изванредан пример. По њима квалитет сервиса[1] је важан чинилац а тешко га је остварити без успоставе веза. Квалитет сервиса је нарочито важан за саобраћај у реалном времену као што су говор и видео.

Ове две међусобно супротстављена правца оличена су у Интернет и ATM[2] мрежама. Интернет нуди транспортном слоју сервис без успоставе везе а ATM нуди сервис са успоставом везе. Без озира на опредељења гаранција квалитета сервиса постаје незаобилазан параметар о коме све мреже морају да воде рачуна па и Интернет.

С обзиром на тип сервиса који се нуди, две различите организације су могуће. Ако је у питању сервис без успоставе везе, пакети се шаљу у подмрежу појединачно и преусмеравају (рутирају) независно од један од другог. Не постоји унапред дефинисано правило или стратегија. Пакети се називају датаграми[3] (аналогија са телеграмима) а подмрежа се назива подмрежа са датаграмима[4]. Ако се користи сервис са успоставом везе, мора се успоставити путања од изворишног до одредишног рутера пре слања пакета. Ова веза се зове виртуелно коло[5] у аналогији са колом које се успоставља у телефонском систему. Одговарајућа подмрежа назива се подмрежа са виртуелним колима[6].

Сервис без успоставе везе

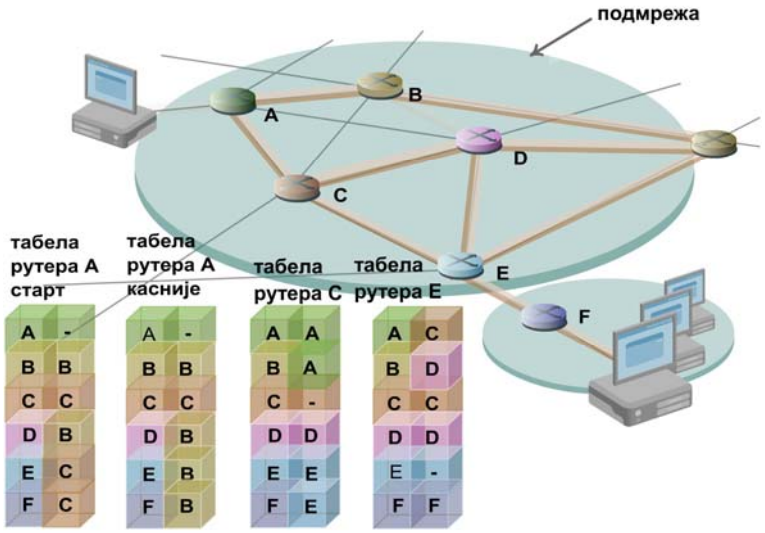
У овом делу анализирћемо како ради подмреже без успоставе везе - са датаграмима (слика 4.2). Претпоставићемо да процес P_1 (станица 1) има дугачку поруку за P_2 (станица 2). Он предаје поруку свом транспортном слоју са успоставом везе да би је он достави процесу P_2. Покреће се извршни кôд транспортног слоја на станици 1 (H_1) који је обично део оперативног система. Додаје се заглавље транспортног слоја испред поруке[7] и као целина се предаје мрежном слоју[8].

---

[1] *Quality of Service*
[2] *Asynchronous Transfer Mode*
[3] *Datagram*
[4] *Datagram subnet*
[5] *Virtual circuit*
[6] *Virtual circuit subnet*
[7] Погледати поглавље о OSI референтном моделу.
[8] Још једна процедура унутар оперативног система.

Ако је порука четири пута дужа од максималне величине пакета, мрежни слој мора да је разбије у четири пакета и пошаље сваки од њих један за другим рутеру А користећи неки од протокола за тачка-тачка везе, нпр. PPP[1]. Сваки рутер има своју табелу записа на основу којих усмерава даље пакете на путу до одредишта. Запис у табели се састоји од адресе одредишта и излазна линија која се користи да би се стигло до одредишта. Могу се користити само директне везе. На пример рутер А (слика 4.2) има само две излазне линије: ка В и ка С. То значи да сваки пакет који стигне мора бити послат једном од ова два рутера, чак и ако је крајње одредиште неки други рутер. Почетна табела за усмеравање рутера А на слици је представљена ознаком „старт".

Када дођу у чвор (рутер) А пакети са редним бројевима 1, 2 и 3 се само кратко задржавају (колико је потребно да се израчуна контролна збир). Затим се усмеравају ка рутеру С, како што је у табели рутера А и записано. Пакет са редним бројем 1 се прослеђује до рутера Е и затим до рутера F. У рутеру F пакет са редним бројем 1 укалупљује се у рам слоја везе[2] и преко локалне рачунарске мреже шаље се станици H_2. Пакети са редним бројевима 2 и 3 следе исту путању.



Слика 4.2. Рутирање у пакета (датаграма) у подмрежи

Када пакет са редним бројем 4 доспе до рутера А, он тај пакет усмерава ка рутеру В без обзира што је одредиште непромењено. Из неког разлога рутер А је одлучио да усмери пакет са редним бројем 4 различитом путањом од оне на коју су усмерена претходна три пакета. Највероватнији разлог да рутер А промени своју табелу рутирања

[1] *Point-to-Point Protocol*

[2] Формира се Етернет (односно IEEE802.3) рам.

је обавештење о загушењу у саобраћају негде дуж путање ACE. Нова путања је на слици 4.2 означена са „касније".

Управљање табелама и доношење одлуке о рутирању назива се алгоритм за рутирање. Њима је у потпуности посвећено једно од следећих поглавља.

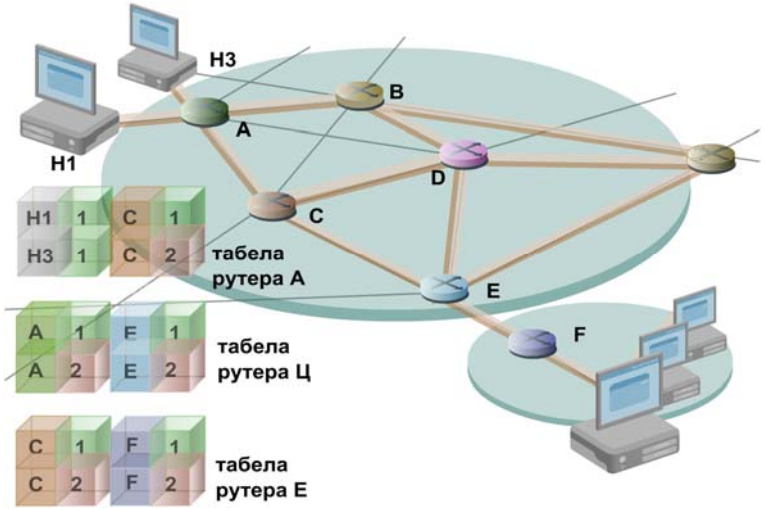
Сервис успоставе везе

За сервис са успоставом везе потребна је подмрежа са виртуелним колима. Концепта виртуелног кола је заснован на идеји да код преусмеравња пакета не треба бирати путању за сваки пакет посебно. Путању од изворишне до одредишне станице (виртуелну везу) треба одабрати у току успостављања везе и записати је у табелама рутера. Она се користи за размену свих пакета (саобраћај) те везе. Принцип је идентичан као у телефонском систему. Кад се веза прекине, виртуелно коло се прекида. Сваки пакет те везе носи идентификатор виртуелне везе (кола) коме припада.

Као пример, сведочи ситуација на слици 4.3. Станица H_1 је успоставио везу са станицом H_2. Први запис у свакој појединачној табели рутера представља ознаку тог виртуелног кола. Прва линија табеле рутера A јасно упућује да пакет са идентификатором везе 1, који шаље H_1, треба проследити рутеру C, задржавајући исти идентификатор везе. Слично, први запис у табели рутера C прослеђује пакет до E, такође са идентификатором1.

Шта се дешава, ако H_3 такође жели да успостави везу са H_2? Изворишна станица опет бира идентификатор 1 и тражи од подмреже да успостави виртуелно коло. Овом приликом посматра се други запис у табелама. Јавља се неусаглашеност. Рутер A лако може да разликује пакете који стижу преко везе 1 које шаље H_1 од пакета везе који стижу преко везе 1, које шаље H_3, али рутер C то не може. Због тога рутер A додељује други идентификатор везе излазном саобраћају. Рутери морају имати могућност замене идентификатора везе у излазним пакетима. Понегде се ово зове замена ознаке[1].

[1] *Label switching*



Слика 4.3. Рутирање у подмрежи са виртуелним колом

Упоређивање сервиса са успоставом и без успоставе везе

У овом одељку анализираћемо предности и мане сервиса са успоставом везе и сервиса без успоставе везе. Сигурно је да ниједан од сервиса не може да задовољи све захтеве. Посматрајмо случај када се преносе пакети мале величине: комплетна одредишна адреса у која би се налазила у сваком пакету који се преноси представља значајно количину некорисних података (премашење[1]) у односу на количину корисних података. Последица је пропусни опсег који се троши бескорисно.

У подмрежи са виртуелним колима пакети садрже идентификаторе везе уместо пуне адресе одредишта. Цена којом се плаћа ова погодност је меморијски простор потребан за смештање записа о вези у табели сваког рутера дуж путање виртуелног кола.

Виртуелна кола захтевају фазу успоставу и раскидање везе што одузима време и троши ресурсе. Са друге стране одлука о томе шта урадити с пристиглим пакетом у подмрежи са виртуелним колом је једноставна и брза: рутер, користећи идентификатор виртуелног кола, проналази у својој табели одговарајући запис о излазној линији преко које усмерава (шаље) даље пакет. У подмрежи са датагрмима потребна процедура претраживања да се пронађе (лоцира) одговарајући запис је сложена и траје знатно дуже.

Још један податак је критеријум за упоређивање: величина меморије која је потребна рутеру за смештање табела. Подмрежа са датаграмом захтева да постоји запис о путањи до сваког одредиште. Подмрежа са виртуелним колом има само запис о сваком виртуелном колу. Ова предност је ипак помало варљива с обзиром на то да се пакети,

[1] *Overhead*

који се размењују у току успоставе везе, такође рутирају. Они садрже пуну изворишну и одредишну адресу као и датаграми.

Виртуелна кола имају значајну предност у гарантовању квалитета сервиса и избегавању загушења у саобраћају пошто се ресурси као што је меморијски простор, пропусни опсег и сл. могу унапред приликом успостављања везе резервисати. Једном кад пакети почну да пристижу преко успостављене везе неопходни пропусни опсег и капацитет рутера биће им на располагању. Код подмреже са датаграмом избегавање загушења је велики проблем.

У системима обрада трансакција (нпр. продавнице које позивају ради провере за куповине кредитном картицом), губитак времена потребног да се успостави и раскине виртуелно коло може лако да умањи предност коју даје поузданост употребљеног кола. Ако се очекује да ће већи део саобраћаја бити оваквог типа употреба виртуелног кола у подмрежи нема смисла.

С друге стране перманентна виртуелна кола која се ручно успостављају и трају месецима или годинама могла би у овом случају да буду од користи .

Виртуелна кола, међутим, имају проблем рањивости. Ако се рутер поквари избришу се сви записи из меморије. Чак и ако се рутер брзо настави са радом, сва виртуелна кола која пролазе кроз њега су изгубљена. Такође када је комуникационе линије у прекиду сва виртуелно коло која користе ту линију су прекинута. Да би се наставила размена пакета између извориштa и одредиштa мора се поново успоставити веза.

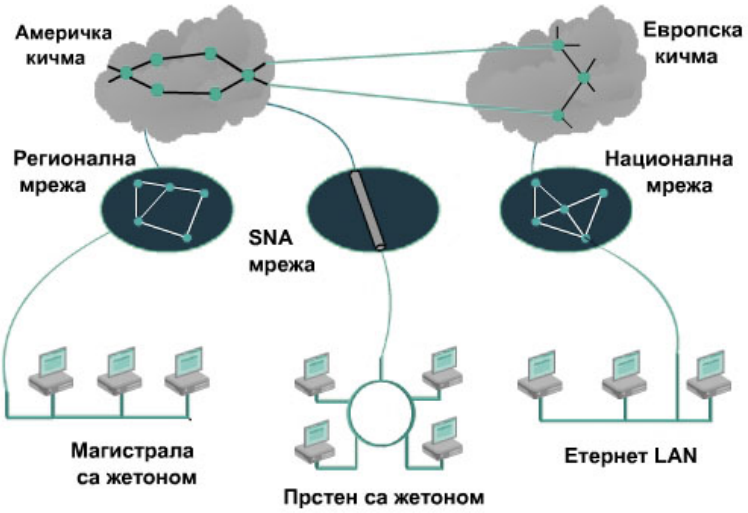
Са друге стране ако се поквари рутер у подмрежи са датаграмима на губитку су само они корисници чији су пакети тада и на том рутеру чекали у реду за обраду. Током слања дугачке секвенце пакета, пошто могу да мењају путању, рутери кроз подмрежу са датаграмима могу и да равномерно расподелељују саобраћај.

4. 1 Мрежни слој на Интернету

Протокол мрежног слоја који се највише користи је Интернет протокол IP[1]. Зато ћемо детаљно анализирати IP протокол као пример протокола мрежног слоја.

На мрежном слоју, Интернет се може видети као колекција подмрежа међусобно повезаних рутерима. Не постоји стварна структура мреже али постоји неколико окосница[2]. Оне се састоје од веза (линкова) великог пропусног опсега и брзих рутера. На главне везе прикључене су регионалне мреже а на њих локалне мреже већих институција, универзитета, компаније и провајдери Интернет сервиса (слика 4.4). Главни везивни елеменат Интернет мреже је IP протокол.

[1] *Internet Protocol*
[2] Кичма, окосница (*backbone*)



Слика 4.4. Интернет је скуп међусобно повезаних мрежа

Основна обележја IP протокола

Протокол IP обезбеђује сервис без успоставе везе између крајњих система. Постоје бројне предности и недостаци оваквог приступа (анализирали смо у претходном одељку). Додаћемо само још неке које су битне за Интернет као глобалну рачунарску мрежу:

- Интернета може да подржи различите типове мрежа од којих су неке без успоставе везе;
- Интернет сервис без успоставе везе је најпогоднији за транспортне протоколе без успоставе везе.

Слика 4.5 приказује:

- уобичајен пример употребе IP протокола: две локалне рачунарске мреже међусобно повезане са мрежом са штафетним[1] преносом[2].
- поступак IP протокола за размену података између станице A из једне локалне рачунарске мреже LAN_1 и станице B из друге локалне рачунарске мреже LAN_2 преко мреже са штафетним преносом.
- архитектуру протокола и формат јединице података у свакој фази. Крајњи системи и рутери морају имати исти IP протокол. Такође, крајњи системи морају

[1] *Frame relay*. Среће се и под називом мрежа за преусмеравње рамова. Спада у доста широко распрострањених технологија за мрежа већих географских растојања WAN (*Wide Area Network*).

[2] Архитектура протокола IEEE 802 се састоји од физичког слоја; слоја контроле везе (MAC) који се односи на адресирање и контролу грешака и слој контроле логичке везе (LLC), који се односи на логичке везе и идентификовање корисника LLC-а.

имати исте протоколе изнад IP протокола. Посредни рутери треба да имплементирају протоколе све до IP протокола.

Са виших слојева целина IP крајњег система A прима блок података који треба да се пошаље до крајњег система B. Додаје заглавље података (у тренутку t_1) које, између осталог, садржи и глобалну Интернет адресу крајњег система B. Та адреса се логички дели на два дела: идентификатор мреже и идентификатор крајњег система. Комбинација IP заглавља и података са вишег слоја је јединица података IP протокола или датаграм. Датаграм се затим пакују:

- у јединице података протокола локалне рачунарске мреже LAN_1 која се састоји од:
  - заглавља LLC подслоја у тренутку t_2,
  - заглавља и краја рама (приколица) MAC подслоја у тренутку t_3.

Затим се шаље рутеру:

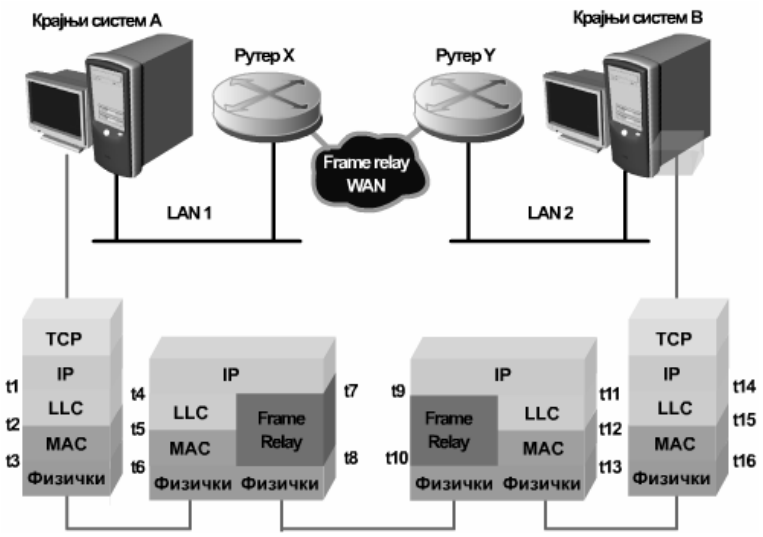
- који скида заглавље локалне рачунарске мреже LAN_1, тј. подслојева LLC и MAC како би прочитао IP заглавље у тренутку t_6,
- поново пакује датаграм са заглављем рама протокола за штафетни пренос у тренутку t_8 и
- прослеђује га преко WAN мреже до другог рутера.

Рутер на другом крају WAN мреже:

- скида заглавље рама протокола за штафетни пренос у тренутку $t_{10,}$
- поново пакује у јединицу података протокола локалне рачунарске мреже LAN_2 која се састоји од:
  - заглавље LLC подслоја у тренутку t_{12},
  - заглавље и краја рама (приколица) MAC подслоја у тренутку t_{13}.
- шаље га крајњем систему B.

Крајњи систем B:

- скида заглавља локалне рачунарске мреже LAN_2, тј. подслојева LLC и MAC како би прочитао IP заглавље у тренутку t_{16},



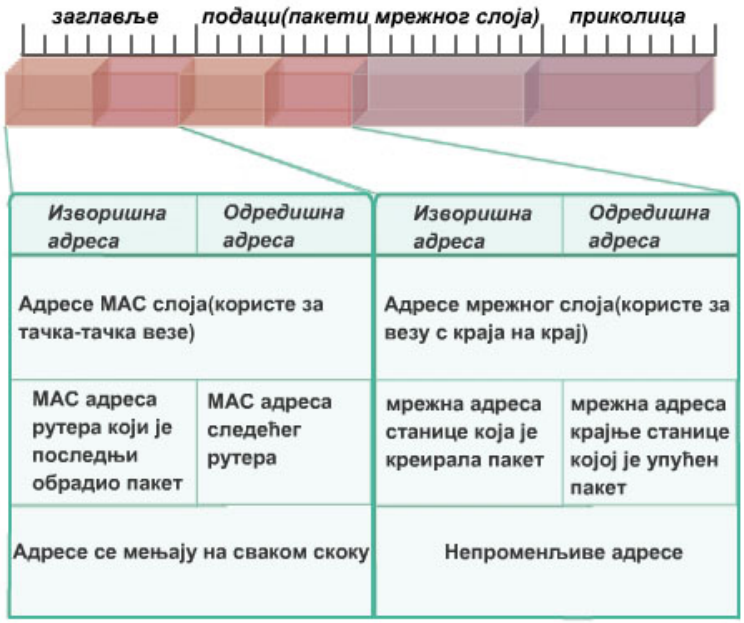
Слика 4.5. Начин рада мрежног протокола на Интернету:IP протокола

Погледајмо овај пример детаљније. Крајњи систем - станица А има пакет који треба да пренесе крајњем систему - станица В; пакет носи IP адресу станице В. Станице А има IP модул који препознаје да је одредиште - станица В у другој мрежи. Први корак је да станица А пошаље податке рутеру, у овом случају рутеру R_1. Да би се то урадило, IP слој прослеђује пакет нижем слоју (у овом случају LLC подслоју) налажући му да га пренесе рутеру R_1. Подслој LLC преноси ове информације MAC подслоју који у своје заглавље на месту одредишне адресе убацује адресу MAC послоја рутера R_1. Сада се рам, који се преноси локалном рачунарском мрежом LAN_1, састоји од: података изнад TCP слоја и заглавља свих слојева кроз које пролази: TCP заглавље, IP заглавље, LLC заглавље, MAC заглавља и приколице[1] (слика 4.5). Пакет се на даље преноси локалном рачунарском мрежом LAN_1 све до рутера R_1. Рутер уклања заглавља MAC и LLC подслојева и анализира IP заглавље да би утврдилио одредиште података у овом случају станица В. Рутер R_1 мора донети одлуку о рутирању. Постоје три могућности:

1. Одредишна станица В (крајњи систем) директно је повезана са једном од мрежа за коју је рутер везан. Ако је тако, рутер шаље пакет директно до одредишта;
2. Да би дошао до одредишта пакет мора да пређе преко једног или више рутера. Значи да рутер R_1 мора донети одлука о рутирању: ком рутера треба послати пакет? У случајевима 1 и 2 IP слој рутера шаље пакет нижем слоју са IP адресом одредишта - станице В (слика 4.6);

[1] Сваки слој везе додаје и почетак (заглавље) рама и крај (приколицу или реп). У даљем тексту употребљаваћемо само термин заглавље једноставности ради али треба имати на уму да се подразумевају оба дела.

3. Рутер не зна одредишну адресу. У овом случају рутер шаље поруку[1] изворишту пакета којом указује да је дошло до грешке.



Слика 4. 6 Структура рама који се преноси кроз мрежу

У овом примеру подаци морају прећу преко рутера R_2 пре него што дођу до одредишта. То значи да рутер R_1 прави нови рам у коме је:

- заглавље слоја везе мреже за штафетни пренос,
- податак је јединица података мрежног слоја (NPDU[2]).

Заглавље мреже са штафетним преносом указује на логичну везу са рутером R_2. Кад овај рам стигне до рутера R_2 заглавље и крај рама се уклањају. Рутер проналази да ова IP јединица податка мрежног слоја упућена станици B који је директно повезан са мрежом за коју је овај рутер везан. Рутер зато прави рам у коме је изворишна адреса другог слоја његова адреса ,одредишна адреса другог слоја је адреса станице B а рам се шаље на локалну рачунарску мрежу LAN_2. Подаци коначно стижу до станице B који уклања заглавља другог и трећег слоја (LLC, MAC и IP).

Пре него што се подаци проследе даље, рутер ће можда морати да пакет подели у мање целине (јединицу података) да би је прилагодио мањој максималној величини пакета одредишне мреже. Свака од новонасталих јединица постаје независна јединица података која се пакује у рам нижег слоја и ставља у ред чекања за даљи пренос. На

[1] *Error message* (порука о грешци). Видећемо у следећим поглављима да постоје специјалне врсте порука.
[2] *Network Protocol Data Unit* или *IP Prorocol Data Unit*

мрежном слоју се те мање целине називају фрагменти а процес деобе се назива фрагментација.

Рутер може да ограничи листу чекања за сваку мрежу за коју је везан како би избегао да спорија мреже успорава бржу мрежу. Када више нема места у реду за чекање, односно када се капацитет попуни новопристигли пакети се једноставно одбацују. Овај процес се понавља у онолико рутера колико их је потребно прећи да би пакет података стигао до одредишта. На исти начин као што то рутер ради одредишни крајњи систем (крајња станица или хост) преузима IP пакет са мреже. Ако је успут извршена фрагментација IP слој одредишног система сакупља пристигле фрагменте и проселеђује их вишем слоју тек када све делове споји у једну целину.

Сервис који нуди IP протокол сматра се непоузданим[1] зато што се не гарантује да ће подаци стићи до одредишта или да ће подаци који стигну бити поређани одговарајућим редом. То је посао вишег слоја (у овом случају транспортног TCP) који води рачуна да крајњи системи разреши грешке настале у преносу[2]. Овакав прилаз обезбеђује велику флексибилност.

Интернет протокол

У овом делу анализираћемо Интернет протокол верзије 4 који се означава са IPv4. Протокол IP је дефинисан документом RFC 791, 950, 919 и 922, 2474. Планира се да ће коначно IPv4 заменити Интернет протокол верзије[3] 6 (IPv6). Интернет IP је део скупа TCP/IP протокола[4] и најшире коришћени протокол за међусобно повезивање рачунарских мрежа. У овом поглављу везано за IP протокол анализираћемо:

- интерфејс са транспортном слоју описујући сервисе које IP обезбеђује,
- проблеме у пројектовању,
- формат и механизам протокола.

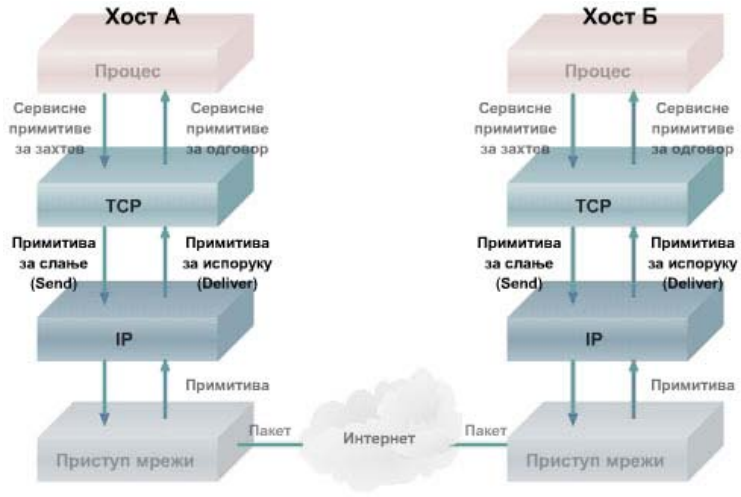
Сервис

Слој IP обезбеђује две примитиве сервиса ка вишем слојевима[5] (слика 4.7) . То су:

- примитива за слање - Send која се користи када се корисник захтева од IP слоја да пренесе податке.
- примитива за испоруку - Deliver која се користи да би IP слој обавестио кориснике да су подаци стигли.

---

[1] У литератури се среће и појам најбољи могући сервис енг. „*best effort*".
[2] Користи се и термин опоравак од грешке енг. „*error recovery*".
[3] Постоји и Интернет протокол верзија 5 - IPv5 али није у јавној употреби.
[4] TCP/IP *Protocol Suite*
[5] Детаљно о примитиваме сервиса описане су у поглављу OSI референтни модел и транспортни слој.



Слика 4.7. Примитиве IP слоја Send и Deliver

Проблеми пројектовања

У овом поглављу анализираћемо неке чиниоце који утичу на пројектовање ситема за IP протоколима:

- рутирање,
- време трајања (животни век) пакета ,
- деоба пакета (фрагментовање) и поновно повезивање,
- контрола грешке,
- контрола тока и
- адресирање.

Време трајања пакета

Ако се користи динамичко или алтернативно рутирање, постоји могућност да пакет у бесконачно дуго кружи Интернетом. Ово је недопустиво из два разлога. Прво, бескрајно дуго кружење пакета троши ресурсе. Друго, транспортни протокол може да зависи од ограничења у трајању пакета. Да би се избегли ови проблеми сваком пакету се додељује време трајања TTL[1] (животни век). Када прође то време пакет бива одбачен.

[1] *Time To Live*

Најједноставнији начин да се реализује трајање пакета је помоћу бројача[1]. Сваки пут када пакет прође кроз рутер бројач се декрементира (умањује за један). Такође, трајање пакета може да означава време. Ово захтева од рутера да зна колико је времена прошло од када је пакет или део тог пакета последњи пут прошао кроз рутер, и да за толико умањи вредност поља TTL. Да би то било могуће потребна је глобална временска сихронизација. Предност коришћења стварне временске величине је што се може користити у алгоритму за поновно повезивање пакета, што ће бити објашњено следећем поглављу.

Деоба и повезивање пакета

Појединачне мреже у Интернету могу назначити различите максималне величине пакета. Не би било препоручљиво условљавати исту величину пакета кроз различите мреже. Према томе рутер мора пристигле пакете, пре него што их проследи следећој мрежи, да дели у мање целине које се називају сегменти или фрагменти.

Ако се пакети могу делити на свом путу кроз мрежу (по могућству више од једном) поставља се питање где ће се они поново спајити. Најлакше решење је да се поновно спајање одради у самом одредишту. Ако би се дозволило поновно спајање у рутерима на међупутањи, могли би да дође до следећих неповољних околности:

- потреба за велики меморијским простором у рутеру. Постоји ризик да се цео меморисјки простор заузме за складиштење делова пакета,
- сви делови пакета морају да прођу кроз исти рутер, што спречава коришћење динамичког рутирања.
- На слоју мреже на Интернету (IP слоју) спајње делова пакета у обавља се у одредишном систему. Код деобе IP пакет из његовог заглавља користе се следеће информације :
- идентификатор пакета ID чија је намена да одреди крајње системе између којих се размењује пакет. Састоји се од изворишне и одредишне IP адресе и броја који указује протокол ког слоја је генерисао податак (нпр. TCP).
- дужина података која представља дужину корисничких података, исказана у октетима (бајтовима),
- одступање које указује на позицију дела корисничких података у пољу података оригиналног пакета исказан као умножак од 64 бита.
- ознака M[2] указује да ли постоји (или не) још делова оригиналног пакета.

Изворишни систем формира пакет[1] такав да је:

[1] Назива се и бројач скокова (*Hop*)
[2] *More*

- вредност поља о дужини података одговара целој дужина поља података,
- одступање има вредност нула,
- ознака М постављена је на вредност нула.

Када IP целина рутера дели пакет велике дужине у два дела она:

- прави два нова пакета и копира заглавље оригиналног пакета у оба дела,
- дели долазеће корисничке податке на два приближно једнака дела водећи рачуна о 64-битној граници. Добијене делове смешта у нове пакета. Први део мора бити умножак од 64 бита,
- поље дужине података првог новог пакета поставља се на вредност једнака дужину уметнутих података. Бит М се поставља на вредност 1. Поље одступање остаје непромењено.
- поље дужине података другог новог пакета поставља се на вредност једнака дужину уметнутих података. Пољу одступање додаје се дужине уметнутих података и првог дела подељеног са 8. Бит М постављен је на вредност 0.

Слика 4.8 описује претходно наведени пример. Описану процедуру је једноставно проширити на уопштену поделу на *n* делова.

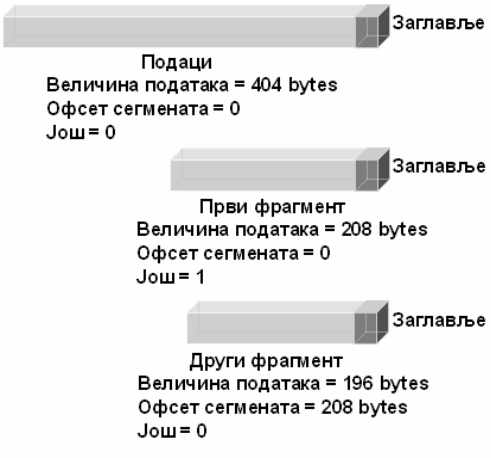
Да би се делови пакета поново повезали у једну целину треба, на месту где се то обавља, да постоји довољно простора у меморији. Како пакети са истим пољем ID пристижу тако се њихови подаци смештају у одговарајући део у меморији све док се цело поље са подацима не повеже у једну целину. Цео процес започиње када у групи пристиглих пакета први од пакета има вредност поља „одступање" једнако нули а последњи од пакета има вредност поља „још" (бит М) постављеним на вредност 0.

Једна од могућности са којом се морамо суочити јесте да један или више делова можда неће стићи до одредишта: IP сервис не гарантује испоруку. Због тога је, како би се ослободило место у меморији, потребан метод којим се одређује када треба одустати од повезивања пакета. Тренутно, користе се два приступа:

- први метод код кога се дефинише време трајања повезивања података када стигне први од делова пакета. Реализује се помоћу локалног часовника реалног времена који је додељен процесу повезивања података. Како се делови пакета смештају у меморију часовник се декрементира. Ако време на који је часовник постављен прође пре него што се заврши склапање података, примљени делови се одбацују;
- други метод користи вредност поља TTL (трајање пакета) који је део заглавља сваког од пристиглих пакета. Процес за склапање података умањује (декрементира) величину овог поља. Као и код првог метода, када поље TTL достигне вредност нула

[1] То је пакет (датаграм) који називамо оригинални пакет.

(истекне време трајања пакета) а сви пакети се не повежу у оригинални пакет, примљени делови се одбацују.



Слика 4.8 Пример деобе пакета (фрагментовања)

Контрола грешке

На Интернету се не гарантује успешну испорука пакета. Када неки од рутера одбаци пакет, требало би да покуша да пошаље неку информацију изворишту пакета. Изворишни протокол Интернет протокол може да искористи ове информације да би промени стратегију слања или можда да обавести више слојеве.

Да би пријавио да је неки пакет одбачен потребна је нешто што би га означило. Пакет може бити одбачен из много разлога, укључујући истицање времена трајања, загушења и грешке на коју указује контролна сума. Када на грешку указује контролна сума, онда није могуће послати информацију о грешци пошто је можда баш поље са изворишном адресом оштећено.

Контрола тока

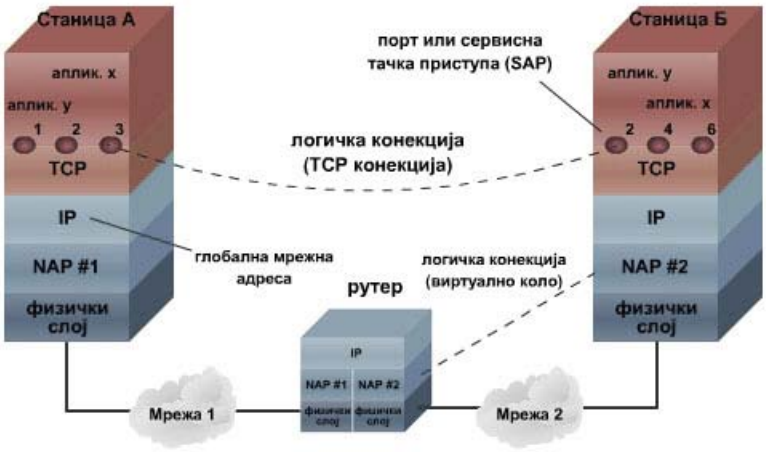
Контрола тока на Интернету дозвољава рутерима и/или одредишним станицама да поставе ограничење у брзини пријема података. За сервиса без успоставе везе који се у овом поглављу анализира, механизам контроле протока је ограничен. Најбољи приступ би био да се другим рутерима и изворишним станицама пошаље пакет за управљање протоком који би захтевао његово смањење. Овакав или сличан пример биће описан касније, када будемо анализирали ICMP протокол[1].

Адресирање

[1] *Internet Control Message Protocol*

Свим системима на крајевима мреже и унутар ње (у међуситемима) додељује се јединствена адреса. У случају TCP/IP архитектуре, ово се односи на IP адресу или једноставно, Интернет адресу. IP адреса се користи за рутирање јединице података мрежног слоја NPDU[1] кроз мрежу или мреже до система на који указује адреса у заглављу јединице података. Адресирања на нижим слојевима је потребно за било коју рачунарску мрежу која има више система коју су у њој међусобно повезана као што је нпр. мрежа са штафетним преносом или ATM мреже. Тада се сваком уређају повезаном на ове мреже додељује јединствена адреса.

Када подаци стигну до одредишног система, они морају бити усмерени ка неком процесу или апликацији тог система. Он ће подржавати више апликација а апликација може подржавати више корисника. Свакој апликацији или сваком кориснику апликације, додељује се јединствени идентификатор, који представља порт у TCP/IP архитектури или приступну тачка сервиса SAP[2] у OSI архитектури[3].



Слика 4.9. Концепт TCP/IP

Свако од поља изворишне и одредишне адресе у IP заглављу садржи 32-битну глобалну Интернет адресу[4] (тзв. IP адресу). Она се састоји од дела који означава мрежу - број мреже[5] и дела који означава крајњу станицу (хоста) у тој мрежи - број станице (хоста)[6]. Комбинација је јединствена: не постоје два уређаја са истом IP адресом.

---

[1] *Network Protocol Data Unit* или IP *Protocol Data Unit*. Уобичајено је да се означавају као датаграми или пакети.

[2] *Service Access Point*

[3] Детаљно је описано у поглавље 2.

[4] Описано је у документу RFC 1166.

[5] *Network number* се користи као термин у документу RFC 1166. У литератури се среће и термин ознака (идентификација) мреже - *NetID* (*Network Identification*).

[6] *Host number* користи се као термин у документу RFC 1166. У литератури се среће и термин ознака (идентификација) крајње станице - *HostID* (*Host Identification*).

Део IP адресе који се односи на мрежу администрира (додељује) једна од три регионалне организације[1]:

- Организација ARIN[2] је одговорна за регистрацију и администрацију IP бројева за северну и јужну Америку.
- Организација RIPE[3] је одговорна за регистрацију и администрацију IP бројева за Европу, Срењи исток и део Африке.
- Организација APNIC[4] је одговорна за регистрацију и администрацију IP бројева за део Азије у области Пацифика.

Интернет адреса, која је 32-битни број, обично се пише у децималној нотацији са тачкама. У овом формату сваки од 4 бајта написан је као децималан од упсегу 0_{10} до 255_{10}. Најнижа IP адреса је 0.0.0.0 а највиша 255.255.255.255. На пример 128.2.7.9 је IP адреса код које је 128.2 број мреже а 7.9 број станице. Правила по коме се IP адреса адреса дели на број мреже и број станице биће објашњена у следећем поглављу. Бинарни формат IP адресе је:

10000000 00000010 00000111 00001001

IP адресе које користи IP протокол јединствено идентификују станицу на Интернету (или уопштено било ком интернету). Ако хоћемо тачно да дефинишемо IP адреса идентификује интерфејс који је у стању да прима и шаље IP датаграме. Један такав систем[5] може да има више таквих интерфејса којим је прикључен на више мрежа има више различитих IP адреса за сваку мрежу. Рутер и крајња станица морају да имају бар једну IP адресу.

IP адресе засноване на класама

Постоји пет класа IP адреса. Уместо равног адресног простора (нпр. 1, 2, 3,...) користи се структура приказана на слици 4.10. Адреса је кодирана да би се дозволиле различите комбинације за специфицирање мреже или станице. Први битови IP адресе указују како ће остали део адресе бити расподељен између дела за мрежу и дела за станицу.

---

[1] Сервис за регистрацију на Интернету (као што су IP адреса и DNS домен) је раније био у надлежности NIC. 1993. год. формиран је InterNIC за све кориснике осим DDN (*Defense Data Network*) на адреси *rs.internic.net*.
[2] *American Registry for Internet Number*.
[3] *Reseaux IP Europeens*.
[4] *Asia Pacific Network Information Centre*.
[5] *Multi-homed*

Слика 4.10. Класе IP адреса

На слици 4.10 види се да:

- Класа A адреса:
  - користи 7 битова за адресу мреже и 24 бита за адресе рачунара (станица). На овај начин може се адресирати $2^7 – 2$ (126) мрежа. У свакој мрежи може да буде $2^{24} – 2$ (16777214) рачунара,
  - опсег адреса је од 0. 0. 0. 0 до 127. 255. 255. 255,
  - одговара мрежама са изузетно великим бројем рачунара.
- Класа B:
  - користи 14 битова за адресу мреже и 16 битова за адресе рачунара (станица). На овај начин може се адресирати $2^{16} – 2$ (16382) мрежа. У свакој мрежи може да буде $2^{24} – 2$ (65534) рачунара,
  - опсег адреса класе B је од 128. 0. 0. 0 до 191. 255. 255. 255,
  - одговара мрежама са релативно великим бројем рачунара.
- Класа C:
  - користи 21 бит за адресу мреже и 8 битова за адресе рачунара (станица). На овај начин може се адресирати $2^{21} – 2$ (2097150) мрежа. У свакој мрежи може да буде $2^8 – 2$ (254) рачунара,
  - опсег адреса класе C је од 192. 0. 0. 0 до 223. 255. 255. 255,

- одговара мрежама са малим бројем рачунара.

- Класа D:
  - користи се за мултикаст (врста бродкаста али за ограничене области и једино за рачунаре који користе исту класу D адреса,
  - опсег адреса је од 224. 0. 0. 0 до 239. 255. 255. 255.
- Класа E:
  - резервисана је за примене у будућности,
  - опсег адреса класе E је од 240. 0. 0. 0 до 247. 255. 255. 255.

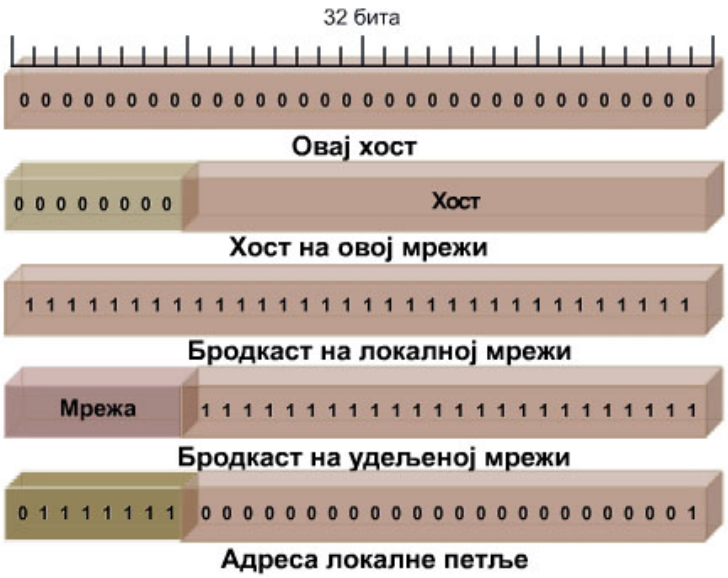
Резервисане IP адресе

Посебне (резервисане) IP адресе добијају се постављањем свих битова на 0 или свих битова на 1 и значења су следећа:

- сви битови 0:
  - у адресном делу рачунара интерпретира се као „овај рачунар“,
  - у адресном делу мреже интерпретира се као „ова мрежа“. Када рачунар хоће да комуницира преко мреже, али још увек не зна мрежни део IP адресе, може да пошаље пакет са <netID>=0 који значи „ова мрежа“. Одговори садрже важећу адресу мреже[1];
- сви битови 1:
  - адреса са свим битовима постављеним на један интерпретира се као све мреже или сви рачунари. На пример адреса 128.2.255.255 значи сви рачунари на мрежи 128.2 (класа B);
- затворена петља[2]:

◊ адреса 127.0.0.1 у класи A је дефинисана као адреса затворене петље. Додељује се интерфејсу за обраду на локалном систему и нема приступ физичкој мрежи.

[1] *Fully qualified network adress*
[2] *Loopback*



Слика 4.11. Неке од посебно резервисаних IP адреса

Станица са више интерфејсних модула имаће више IP адреса - за сваки интерфејс по једну. Додељивање адреса рачунарима (станицама) надлежност је администратора система.

Подмреже

Као последица наглог развоја Интернета принцип додељивања IP адресе постао је превише сложен да би омогућио брзе промене конфигурација локалних мрежа. До промене може да дође када се:

- нови тип физичке мреже инсталира на тој локацији,
- значајно повећа број станица па се захтева деоба локалне мреже у две или више одвојених подмрежа,
- повећање раздаљина која захтева деобу мреже у више мањих подмрежа које су повезане уређајима за међусобно повезивање (гејтвеја).

Да би се спречило додавање мрежа са новим IP адресама од свих рачунара захтева се да подржавају адресирање подмрежа[1]. Додељивање адреса је локално. Мрежа са подмрежама[2] је споља виђена и даље као једна IP мрежа. Уместо да се посматра само

[1] Дефинисано је документом RFC 950

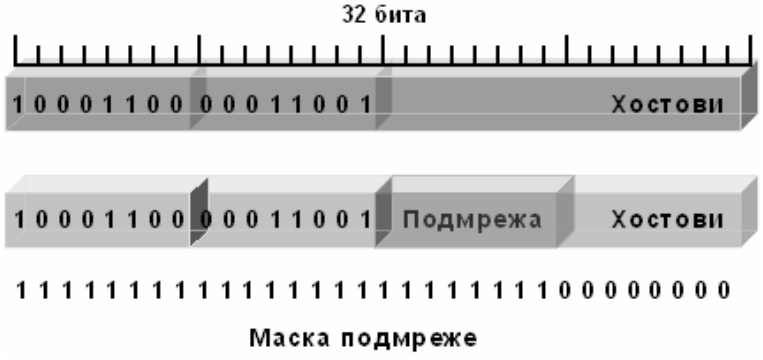
[2] Овај појам је у конфликту са појмом „подмрежа", што значи скуп рутера и комуникационих линија у мрежи (као што смо у уводу овог поглавља поменули). Из контекса се може, највероватније, видети на који појам се односи.

адреса мреже и адреса станице (у оквиру IP адреса) део адресе станице дели се у два дела: у адресу подмреже и адресу станице. Ова идеја има смисла пошто адресе у класи А и класи В имају превише битова додељених адресама рачунара (станица): 2^{24}-2 (А), 2^{16}-2 (В). У пракси није уобичајено да се толико рачунара повезује у једну мрежу.

По добијању IP адресе за мрежу од овлашћене организације, ствар је локалног администратора система да ли да прави подмреже или не. Уколико прави подмреже онда је питање колико битова се додељује за адресирање подмреже а колико за адресирање рачунара (станице). Да би се одредило који део IP адресе представља адресу мреже а који адресу станице користи се маска подмреже[1]. Маска је тридесет двобитна вредност у којој су (слика 4.12):

- на 1 постављене све позиције у мрежном делу адресе [2],
- на 1 постављене све позиције у подмрежном делу адресе[3],
- на 0 постављене све позиције у делу адресе за рачунар[4].

Унутар подељене мреже, локални рутери морају рутирати на бази проширеног мрежног броја који се састоји од мрежног дела IP адресе и броја подмреже. Коришћење маске подмреже помаже станици да утврди да ли је пакет намењен станици на истој локалној мрежи (шаље се директно) или на другој локалној мрежи (шаље се рутеру). Усвојено је да се неки други начини (нпр. ручна конфигурација) користе да би се маска подмреже направила и да би била позната локалним рутерима.

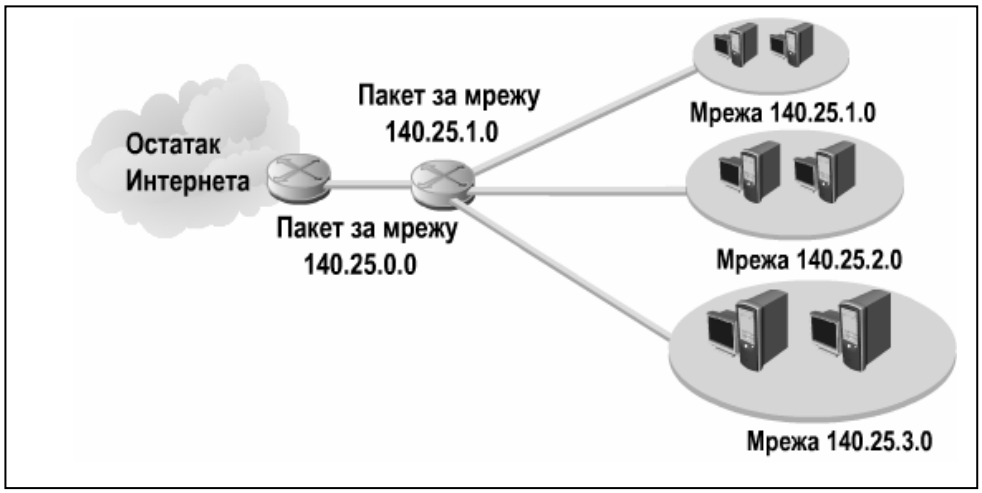


Слика 4.12. Подмрежа и маска подмреже

Комбинација броја подмреже и броја станице назива се и локална адреса или локални део IP адресе. Подмрежавање се примењује тако да је невидљиво (транспарентно) за удаљене мреже (сл. 4.13). Структура подмреже рачунару у оквиру мреже, која је подмрежена, је позната. Рачунару у другој мрежи је структура подмреже

[1] *Subnet mask*
[2] *NetID*
[3] *SubbnetID*
[4] *HostID*

непозната. Он и даље третира локални део IP адресе као адресу рачунара. Деобу локалног дела IP адресе на адресу подмреже и адресу станице бира локални администратор. Било који битови у локалном делу се могу користири за прављење подмрежа.



Слика 4.13. Пример коришћења подмрежа

Слика 4.13 показује пример коришћења подмрежа. Приказан је локални комплекс који се састоји се од три локалне рачунарске мреже LAN_1, LAN_2 и LAN_3 и два рутера R_I и R_K. Остатак Интернета овај комплекс (везан преко рутера R_I) види само као мрежу класе B са мрежном адресом 140.25.x.x где су лева два октета број мреже а десна два октета број станице. Рутер R_K који дели мрежу на подмреже конфигурисан је маском подмреже која има вредност 255.255.255.0. На пример, ако датаграм са одредишном адресом 140.25.2.1 стигне у рутер P_K са остатка Интернета, рутер користи маску подмреже да би утврдио да се ова адреса односи на подмрежу LAN_1 и онда прослеђује пакет тој под мрежи где свака станица (хост) онда одређује да ли је њој намењен пакет.

Постоје два начина подмрежавања:

1. Статичко које подразумева да све подмреже добијене деобеом исте мреже користе исту маску подмреже. Овај начин је једноставан за имплементацију и одржавање. Недостатак је што у случају малих мрежа неке од адреса остају неискоришћене. Узмимо за пример мрежу са четири станице која користи маску подмреже 255.255.255.0. Оваквом доделом остаје 250 IP адреса неискоришћено. Од свих станица и рутера очекује се да подржавају статичко подмрежавање.

2. Код подмрежавања променљиве дужине, подмреже добијене деобом исте мреже могу да користе маске подмреже које се међусобно разликују. Подмрежа са малим бројем станица може да користи маску подмреже одговарајуће величине. Подмрежа са великим бројем станица захтеваја другачију маску. Могућност да се доделе маске подмреже према потребама појединих подмрежа доприноси да се IP адресе беспотребно не расипају.

Употреба маске подмреже

Када се користи подмрежавање сама класа IP адресе не може да одреди који део адресе представља адресу мреже а који адресу станице.

На слици 4.14 приказане су маске подмреже у класи B када је за подмрежу одвојено 8 и 10 битова.

	16 битова	8 битова	8 битова
Класа B	Адреса мреже	адреса подмреже	адреса станице
Маска подмреже	1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1	1 1 1 1 1 1 1 1	0 0 0 0 0 0 0 0
Децимално	255.255.255.0		
Хексадец.	f f f f f f 0 0		
	16 битова	10 битова	6 битова
Класа B	Адреса мреже	Адреса подмреже	Адреса станице
Маска подмреже	1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1	1 1 1 1 1 1 1 1 0 0	0 0 0 0 0 0
Децимално	255.255.255.192		
Хексадец.	f f f f f f c 0		

Слика 4.14. Подмреже у класи B са подмрежама од 8 и 10 битова

Иако се IP адресе нормално означава у децималној нотацији, маска подмреже често се пише у хексадецималној, нарочито када границе нису границе бајтова.

Када рачунар поседује IP адресу и маску подмреже, он може да закључи да ли је пакет упућен ка:

- рачунару у истој мрежи у којој је и он,
- рачунару на другој подмрежи у оквиру исте мреже,
- рачунару на другој мрежи.

Познавање сопствене IP адресе указује да ли је у питању класа A, B или C (на основу битова највеће тежине) тј. где је граница између мрежног и подмрежног дела адресе. Маска подмреже указује где се налази граница између дела адресе подмреже и дела адресе рачунара. Примерима који следе (1, 2, 3) прецизно се осликава употреба маски подмреже. Размотрићемо три различита случаја одредишне IP адресе:

- Уколико је одредишна IP адреса 140.25.1.5 видимо да је мрежни део адресе исти (140.25), али је део за подмрежу различит (1 и 4). Значи да се рачунари налазе на различитим подмрежама и рутер R_K ће га усмерити на одговарајућу подмрежу (слика 4.13). На слици 4.15 приказано је како се употребом маске подмреже обавља упоређивање две IP адресе.
- Уколико је одредишна IP адреса 140.25.1.22 видимо да је мрежни део адресе исти (140.25) и део адресе за подмрежу је исти (1). Део адресе за рачунар је наравно различит (5 и 22). Значи да се рачунари (изворишни и одредишни) налазе на истим мрежама па га рутер R_K неће преусмеравати (слика 4.13);
- Уколико је одредишна IP адреса 192.168.1.225 (адреса из класе C) мрежни делови адреса су различити. Одредишни рачунар не припада ниједној од подмрежа које су у надлежности рутера R_K. Рутер R_K ће овај пакет да проследи рутеру R_I који ће на основу табеле рутирања да усмерава овај пакет даље кроз Интернет.

Пример 1: Претпоставимо да је изворишна IP адреса рачунара 142.25.1.1 (класа B) и маска подмреже 255.255.255.0 (8 битова за адресу подмреже и 8 битова за адресу рачунара).

	Адресе мрежа су исте		Адресе подмрежа нису исте	
	16 битова		8 битова	8 битова
Класа B	140	25	1	5
Маска подмреже	1 1 1 1 1 1 1 1	1 1 1 1 1 1 1 1	1 1 1 1 1 1 1 1	0 0 0 0 0 0 0 0
	255.255.255.0			
Класа B	140	25	4	22

	Адресе мрежа су исте		Адресе подмрежа су исте	
	16 битова		8 битова	8 битова
Класа B	140	25	1	5
Маска подмреже	1 1 1 1 1 1 1 1	1 1 1 1 1 1 1 1	1 1 1 1 1 1 1 1	0 0 0 0 0 0 0 0
	255.255.255.0			
Класа B	140	25	1	22

Слика 4.15. Поређење две адресе класе B

Пример 2: Станица са адресом 192.168.1.65 и маском подмреже 255.255.255.224 треба да пошаље пакет на адресу 192.168.1.91.

Моја адреса	11000000	10101000	00000001	01000001
Маска	11111111	11111111	11111111	11100000
И (AND)				

Моја мрежа	11000000	10101000	00000001	01000000
Одредишна адреса	11000000	10101000	00000001	01011011
Маска	11111111	11111111	11111111	11100000
И (AND)				
Одредишна мрежа	11000000	10101000	00000001	01000000

Одредиште је на истој мрежи као и рачунар (станица) који шаље пакет.

Пример 3: Станица са адресом 192.168.1.65 и маском подмреже 255.255.255.224 треба да пошаље пакет на адресу 192.168.1.97.

Моја адреса	11000000	10101000	00000001	01000001
Маска	11111111	11111111	11111111	11100000
И (AND)				
Моја мрежа	11000000	10101000	00000001	01000000
Одредишна адреса	11000000	10101000	00000001	01100001
Маска	11111111	11111111	11111111	11100000
И (AND)				
Одредишна мрежа	11000000	10101000	00000001	01100000

Одредиште није на истој мрежи као и рачунар (станица) који шаље пакет.

Променљива дужина дела за подмрежу VLS[1]

Оригинална имплементација подмрежа подразумева да се мрежа подели на сегменте исте величине. Овакав начин поделе адресног простора не одговара увек стварним потребама. Променљива дужина дела за подмрежу VLS омогућава креирање сегмената различите величине. У суштини врши се даље подмрежавање подмрежа на претходно описан начин. Систем VLS треба да буде подржан од стране протокола за динамичко рутирање. Већина нових рутера има подршку за VLS. Уобичајено је да рачунари чувају маску подмреже у конфигурационом фајлу.

Да би смо јасније представили потребе за VLS системом посматрајмо корпорацијску мрежу којој је додељена адреса из класе C: 165.214.32.0. Захтева се да се адресни простор раздвоји у пет независних мрежа таквих да:

- подмрежа број 1 има 50 рачунара,
- подмрежа број 2 има 50 рачунара,
- подмрежа број 3 има 50 рачунара,
- подмрежа број 4 има 30 рачунара,

[1] *Variable Length Subnetting*

- подмрежа број 5 има 30 рачунара.

Овај задатак се не може разрешити статичким подмрежавањем. У овом примеру статичко прављење подмрежа дало би четири подмреже (2^2) од којих свака може да има 62 ($2^6 – 2$) рачунара или 8 (2^3) подмрежа од којих би свака имала по 30 ($2^5 – 2$) рачунара. Ниједно од понуђених решења не задовољава постављене захтеве. Да би се мрежа поделила у пет подмрежа треба дефинисати две различите маске подмрежа:

- маском 255.255.255.192 мрежу можемо поделити у четири подмреже са по 62 рачунара,
- четврту подмрежу можемо даље да поделимо у две подмреже са по 30 рачунара користећи маску 255.255.255.224.
- Добили смо три мреже са по 62 рачунара и две мреже са по 30 рачунара. Ово решење задовољава постављене захтеве.

Рутирање

Важна функција мрежног слоја и IP слоја као његовог најраспрострањенијег представника је рутирање. Један од начина поделе рутирања је на основу међусобне позиције изворишне и одредишне станице. Постоје две врсте рутирања: директно и индиректно.

Директно и индиректно рутирање

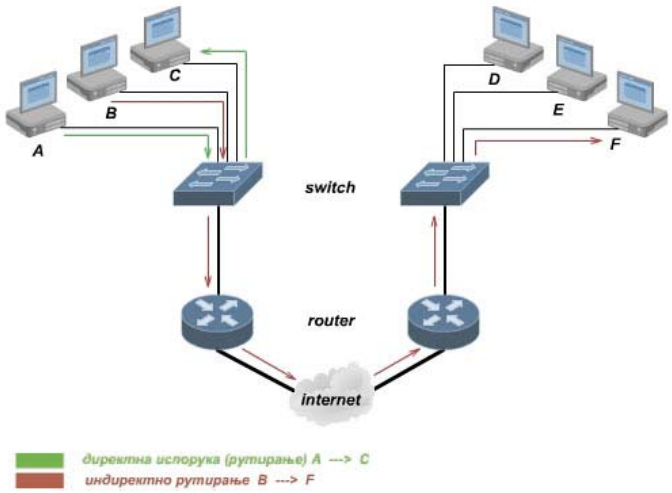
Уколико је одредишни станица повезан на исту физичку мрежу као изворишна станица IP пакети се могу директно размењивати. Реализује се паковањем IP пакета у рам физичке мреже за коју су обе станице везане[1]. Користе се и термини директна испорука или директно рутирање.

Индиректно рутирање се обавља када одредишна станица није повезана на исту физичку мрежу на коју је повезана изворишна станица[2]. Једини начин да једна станица стигне до друге станице је преко рутера (мрежних излаза - гејтвеја)[3]. Адреса првог рутера у алгоритмима за рутирање IP пакета назива се индиректна рута. Адреса првог рутера једина је информација која је потребна изворишној станици да би послала пакет ка одредишној станици.

За комуникацију пара станица када постоји више подмрежа у оквиру исте физичке мреже користи се индиректно рутирање. Рутер је неопходан за преусмеравање саобраћаја између подмрежа (слика 4.16).

---

[1] Погледати објашњење уз слику 4.5.
[2] Станице A и B на слици 4.5.
[3] Треба уочити да се у TCP/IP окружењу рутер и мрежни излаз - гејтвеј користе у истом контексту.



Слика 4.16. Директно и индиректно рутирање

Рутирање

Директне путање (руте) се проналази на основу листе локалних интерфејса. Она се аутоматски формира у процесу иницијализације IP рутирања. Поред тога листа мрежа и придружених им рутера (за индиректно рутирање) може се и конфигурисати. Свака станица садржи податке о вези између:

- одредишне IPадресе и
- путање до следећег рутера.

Информације су смештене у табели која се назива табела рутирања. Могу се срести три врсте међусобних веза (мапирања):

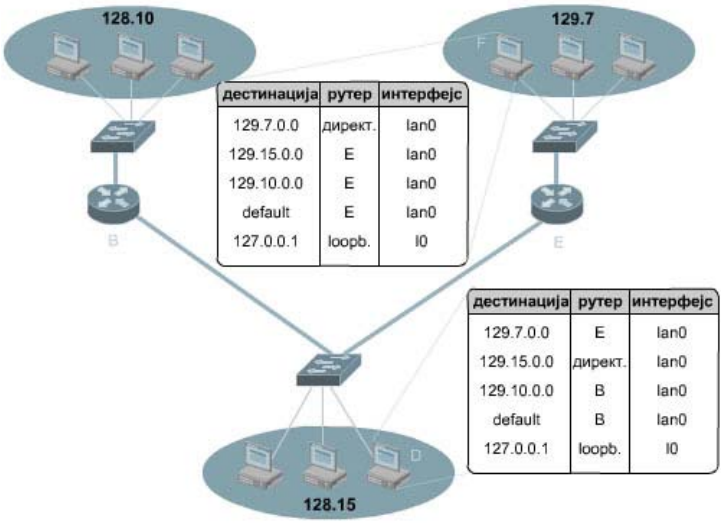
- директне путање за интерфејсе који су повезани на исту физичку мрежу,
- индиректне путање које указују на мреже до којих се може доћи само преко једног или више рутера,
- подразумевана путања која садржи (директну или индиректну) путању ако одредишна мрежа није пронађена на један од претходна два начина.

На слици 4.17 приказан је пример мреже. Табела рутирања станице D може да садржи симболичне записе као што се може видети у табели 4.1.

Одредиште	Путања	Интерфејс
129.7.0.0	E	LAN0
128.15.0.0	D	LAN0

128.10.0.0	B	LAN0
подразумевана	B	LAN0
127.0.0.1	петља	10

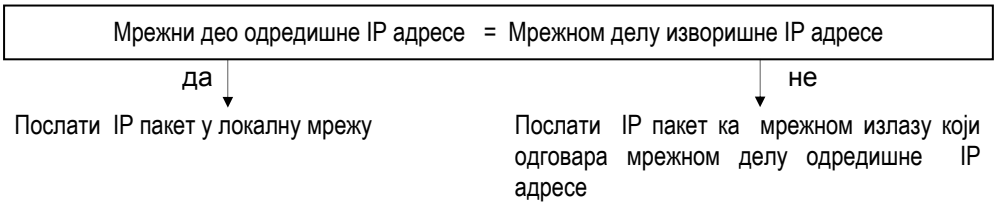
Табела 4.1 Садржај табеле рутирања станице D



Слика 4.17. Сценарио за рутирање помоћу табеле рутирања

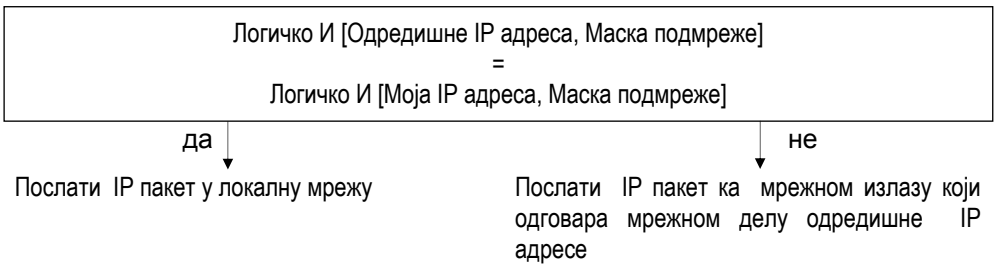
Пошто је станица D директно повезан на мрежу 128.15.0.0 он има директну путању ка овој мрежи. Да би станица D дошла до мреже 129.7.0.0 и 128.10.0.0 мора да користи индиректну путању преко рутера E и B пошто за те мреже станица D није директно везана.

Доношење одлуке о рутирању може се шематски приказати као на слици 4.18.



Слика 4.18. Рутирање без подмрежа

Када је потребно донети одлуку о рутирању пакете између подмрежа онда се то може шематски приказати као на слици 4.19.



Слика 4.19. Рутирање са подмрежавањем

Последице измена алггоритма представљеног на слици 4.18 алгоритмом представљеним на слици 4.19 су следеће:

- Алгоритам на слици 19 представља генерално промену алгоритма за рутирање. То значи да рутери (мрежни излази - гејтевеји) морају да раде са новим алгоритмом;
- Рутирање IP пакета примењује се у свакој станици (а не само у рутерима). Све станице у подмрежи морају да имају:
  - алгоритме за рутирање који подржавају подмрежавање,
  - исту маску подмреже;
- Уколико станица не подржава подмреже онда ће она моћи да комуницира са свим станицама те подмреже али неће моћи да комуницира са рачунарима других подмрежа те мреже. Разлог је станица види све као једну мрежу и не може да направи разлику између станице подмреже и мрежног излаза (рутера) те подмреже[1].

Табеле рутирања могу бити статичке или динамичке. Статичка табела може да садржи алтернативне путање у случају да одређени рутер није доступан. Динамичка табела је флексибилнија у случајевима када дође до нагомилавања пакета или грешкака у преносу. Узмимо на пример Интернет: ако један рутер прекине са радом сви његови суседи шаљу статусни извештај који онда омогућава рутерима и станицама да ажурирају своје табеле рутирања. Сличан план се може користити и за управљањем загушења у саобраћају (нагомилавања пакета).

Табеле рутирања могу се користити и као помоћ осталим сервисима међусобно повезаних мрежа као што су сигурност и приоритети. На пример, појединачне мреже могу служити за преузимање података до одређене сигурносне границе. Механизам рутирања мора обезбедити да ће подацима траженог сигурносног нивоа бити забрањено да пролазе кроз мреже које нису овлашћене да преузимају такве податке.

Једна од техника рутирања је и тзв. изворишно рутирање[2]. Изворишна станица одређује путању тако што у пакету који шаље наводи низ узастопних рутера преко којих

[1] Уколико већи број станица не подржава IP адресе са подмрежама постоји додатно решење - *proxy ARP*.
[2] *Source routing*

пакет треба да пређе. Ова техника може бити корисна када се захтева сигурност и одређени приоритети.

Један од сервиса који је везан за рутирање је и записивање путање (руте). Да би се записала путања сваки од рутер преко кога пакет пређе може да у одговарајућу листу дода и своју IP адресу. Сервис се може употребити код тестирања или отклањања грешака у преносу.

Методе испоруке пакета

Највећи број IP адреса користи се за једног примаоца или метод испоруке ка једној станици - уникаст[1].

Поред овог метода који представља једно извориште - једно одредиште постоје још три метода испоруке[2] када се IP адреса користи за адресирање више станица:

- IP адреса упућена свима - бродкаст адреса[3],
- IP адреса упућена групи станица - мултикаст адреса,[4]
- IP адреса упућена једној од станица групе - еникаст адреса[5].

На слици 4.20 приказана су поменута четири метода испоруке пакета односно врсте адреса. На пример један управљачки центар у мрежи треба да обавести све кориснике да има проблема у раду (мрежа пада). Адреса за више примаоца може бити бродкаст намењена свим станицам одређеног домена или мултикаст намењена станицама одређене подмреже или некој групи.

Адресе један ка свима (бродкаст) никада се не могу употребити као изворишне адресе. Постоје различити типови ових адреса:

- адресу ограниченог дејства (децимално представљена је 255.255.255.255 тј. све јединице) препознаје свака станица а рутери не преусмеравају пакете са овом адресом[6].
- адреса један ка свима одређене мреже је адреса којој је мрежни део адресе одговара мрежи у којој се размењује пакети а део за крајње станице је постављен на све јединице (на пример 128.2.255.255). Ова адреса се односи на

---

[1] *Unicast addresss*

[2] Користи се и термин тип или врста адреса.

[3] *Broadcast addresss*

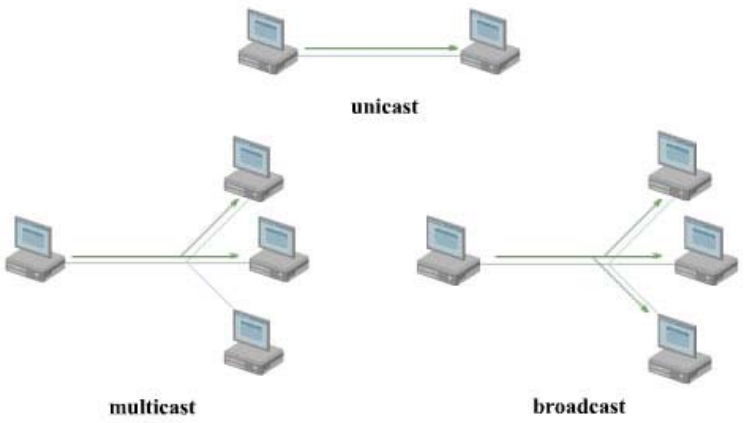
[4] *Multicast addresss*

[5] *Anycast*

[6] Изузетак представља правило BOOTP *forwarding* (преусмеравање). Протокол BOOTP користи адресу 255.255.255.255 да би станицама без диска омогућио успоставу везе са сервером. Када рутери не би преусмеравли ове пакет било би потребно да се у сваку од мрежа постави по један сервер.

све станице у специфицираној мрежи. Рутери би требало да преусмеравају ове пакете[1].

- адреса један ка свим одређене подмреже - ако је мрежни део адресе важећа адреса и ако је део адресе за подмрежу важећи а део за станицу је постављен на све јединице онда је пакет са таквом адресом упућен свим станицама те (специфициране) подмреже. Пошто је маска подмреже изворишне станице различита од маске одредишне станице извориште мора да нађе маску одредишне станице. Рутери усмеравају ове пакете.
- адреса упућена свим подмрежама - ако је мрежни део адресе важећи и ако је део за подмреже постављен на све јединице (нпр. 128.2.255.255) онда је пакет са том адресом упућен ка свим станицама свих подмрежа те (специфициране) мреже. Рутери би могли да преусмеравају пакете са оваквом адресом[2] али у пракси то не раде.



Слика 4.20. Методе испоруке IP пакета

Постоји потреба да се пакет испоручи само одређеном броју станица. У ту сврху користи се класа D адреса и метод испоруке који се означава као мултикастинг. Станице којима се испоручује пакет означавају се као група станица[3]. Пакети са мултикаст адресама испоручују се само члановима групе.

IP адреса упућена једној од станица групе може се описати на следећи начин: једна еникаст адреса се додељује групи станица. Пакет који се пошаље на ту адресу усмерава

[1] Односи се на ARP захтев. Биће обрађено у делу о ARP протоколу.
[2] У случају када преусмеравају користе агоритам *Reverse path forwarding* да би спречили неконтролисано повећавање броја пакета. У документу RFC 922 може се наћи више детаља о овом алгоритму.
[3] *Host group*

се до најближе станице која има ту еникаст адресу. Протокол за рутирање по својим критеријумима одређује најближу станицу.

Понекад један IP сервис може да понуди више станица. На пример корисник жели да учита податке користећи FTP сервис. Подаци се налазе на више FTP сервера и могу се скинути са сваког од њих. Све станице које могу да обезебеде FTP сервис поседују еникаст адресу. Веза се успоставља са станицом из групе станица са еникаст адресом која прва одговори. На тај начин се гарантује да је добијена сервис из најбоље везе станице и пријемника.

Приватне IP адресе

Један од метода да се сачува IP адресни простор је додељивање адреса у приватним мрежама - интранету[1]. Правило да је IP адреса глобална адресе на тај начин је прекршено. За мреже које не захтевају повезивање на Интернет резервисан је део адресног простора који оне користе. Уобичајено је да је администрација интранета у надлежности једне организације. Опсег адреса је:

- 10.0.0.0 једна мрежа класе A,
- 172.16.0.0 до 172.31.0.0 - укупно 16 узастопних мрежа са адресама из класе B,
- 192.168.0.0 до 192.168.255.0 - укупно 256 узастопних мрежа са адресама из класе C.

Било која организација може да користи било коју од адреса из горе поменутих опсега. Ове адресе се не размењују са рутерима ван приватних мрежа (екстерним рутерима).

Рутирање са IP адресама које нису засноване на класама[2]

Стандардно IP рутирање познаје само адресе класа A, B и C. Највише је искоришћен адресни простор класе B. Као следећи најпогоднији показао се адресни простор класе C. Тако је генерисан нови проблем: превелики број записа у табелама рутирања потребних за сваку од мрежа из класе C. На пример за 3000 станица класе B потребан је само један запис у табели рутера окоснице Интернета. За исти број станица класе C потребно је 16 записа.

Проблем се може решити са IP адресама које нису засноване на класама - CIDR рутирањем. Среће се и под називом супер умрежавање[3] и описан је у документима RFC 1518, 1519, 1467.

Основна идеја CIDR адресирања је доделити више IP адреса на такав начин да обезбеђује здруживање у мањи број записа у табелама рутирања. На пример ако је једној

---

[1] Описан у документу RFC 1918 *Address Allocation for Private Internets*.

[2] CIDR (*Classless Inter Domain Routing*) - рутирање између домена са IP адресама које нису засноване на класама.

[3] *Supernetting*

локацији (сајту) додељено 16 адреса из класе C и њих 16 је на таквом месту да се могу здружити онда се на сваку од њих може односити само један запис у табели рутирања рутера на Интернету. Такође ако је 8 различитих локација повезано преко једне прикључне тачке на истог добављача Интернет услуга[1] и свакој од локација је додељено 8 различитих IP адреса које се могу здружити онда је потребан само један запис у табелама рутирања за све њих. Да би се могло извршити здруживање потребне су три ствари:

1. Да би се могле заједно рутирати IP адресе које треба здружити морају да имају исте битове највеће тежине;
2. Табеле рутирања и алгоритми за рутирање морају своје одлуке доносити на основу 32-битне IP адресе и 32-битне маске;
3. Протоколи за рутирање који се користе морају размењивати поред адресе и 32-битну маску.

Свака CIDR табела рутирања садржи 32-битни IP адресу и 32-битну мрежну маску које заједно дају дужину и вредност IP префикса. Ово се представља као пар <IP адреса, мрежна маска>. На пример за адресирање блока од осам адреса из класе C са једном једином табелом рутирање добијамо:

<192.32.136.0 255.255.248.0>

Са гледишта окоснице мреже ово се може реферисати као једна мрежа класе C у опсегу 192.32.136.0 до 192.32.143.0. Ово је приказано на слици 4.21.

192.32.136.0 класа C	11000000	00100000	10001000	00000000
255.255.248.0	11111111	11111111	11111000	00000000
Маска				
И (AND)				
192.32.136	11000000	00100000	10001000	00000000
IP префикс				
192.32.143.0 класа C	11000000	00101000	10001111	00000000
255.255.248.0	11111111	11111111	11111000	00000000
Маска				
И (AND)				
192.32.136	11000000	00101000	10001000	01000000
IP префикс				

Слика 4.21. Пример CIDR супер умрежавања

Формат и механизам протокола

[1] Интернет провајдер.

Протокол између IP целина најбоље се може описати посматрањем формата IP датаграм (пакета) који се састоји од два дела: заглавље и дела за податке. Заглавље чине:

- део дужине 20 бајтова тачно дефинисаних садржаја поља и
- део променљиве дужине које је опциони.

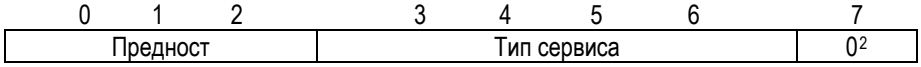
Формат заглавља је приказан на слици 4.22. Он се шаље редоследом слева на десно а на првом месту је бит највеће тежине поља „верзија”[1] . Поља су постављена следећим редоследом:

Слика 4.22. Формат заглавља IPv4

- Верзија (дужине 4 бита) носи информацију којој верзији протокола пакет припада. Укључујући верзију у сваки пакет, постаје могуће обезбедити прелаз између рачунара који раде са старим верзијама протокола и другим које раде са новим. Вредност овог поља је 4 код IPv4.
- Дужина заглавља IHL[2] (дужине 4 бита) није константна тако да ово поље у заглављу обезбеђује информацију колико је заглавље дугачко изражено у 32-битној речи. Минимална вредност је 5 за минималну дужину заглавља од 20 бајтова која се примењује када опције не постоје у заглављу. Максимална вредност овог 4-битног поља је 15. Тиме је ограничено заглавље на 60 бајтова што значи да је максимална величина поља опција 40 бајтова. За неке опције, као што су оне које праве запис о рути (путањи) коју пакет пређе, 40 бајтова је превише мало, чинећи опцију бескорисном.

[1] *Version*
[2] *Internet Header Length*

- Тип сервиса[1] (дужине 8 бита): дозвољава станици да укаже подмрежи коју врсту сервиса (тј. који квалитет сервиса) жели за овај пакет. Могуће су различите комбинације поузданости и брзине. За дигитализовани говор, брза испорука пакета је важнија од тачне испоруке. За пренос фајлова важнији је пренос без грешке него брз пренос. Поље садржи информације приказане на сл. 4.23.



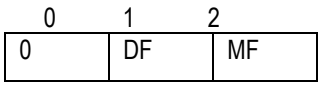
Слика 4.23. Тип сервиса

  - Предност[3] је поље које указује какав је пакет и какав приоритет му треба доделити. Постоје следеће могућности:
    - ✓ 000 уобичајени приоритет,
    - ✓ 001 - 101 различите категорије приоритета,
    - ✓ управљање је у надлежности међумреже,
    - ✓ управљање је у надлежности мреже.
  - Тип сервиса [4] има следеће могућности:
    - ✓ 1000 да смањи кашњење на најмању могућу вредност,
    - ✓ 0100 да повећа пропусну моћ што је више могуће,
    - ✓ 0010 да омогући што већу поузданост,
    - ✓ 0001 да обезбеди најмање трошкове,
    - ✓ 0000 нормални сервис.
- Укупна дужина[5] (дужине 16 бита) показује колика је дужину пакета: заглавља и података изражено у бајтовима. Максимална дужина је 65 535 бајтова. За сада је горња граница прихватљива али са гигабитским мрежама потребни су већи пакети;
- Идентификација[6] (дужине 16 бита) је потребна да би омогућила удаљеној станици да одреди ком пакету управо доспели део припада. Сви делови добијени деобом једног пакета садрже исту идентификациону вредност;

---

[1] У литератури (Столингс) ово поље означавају као DS/ECN (*Differentiated Services* - врсте сервиса)/(*Explicit Congestion Notification* - обавештење о загушењу) што ће бити објашњено детаљније у једном од следећих поглавља. Ми ћемо се овде придржавати ознака и објашњења датих у документима RFC760 и RFC1349.
[2] MBZ - *Must Be Zero* (мора бити нула се на нулу) - резервисано за будућу употребу
[3] *Precedence*
[4] TOS - *Type Of Service*
[5] *Total Length*
[6] *Identification*

- Ознаке (флегови) (дужине 3 бита): ово поље садржи ознака за управљање од којих су само два тренутно дефинисана (слика 4.24):



Слика 4.24. Ознаке (флегови) у IP датаграму

  - Први бит за сада мора да буде 0;
  - Други бит означен са DF[1]:
    - ✓ ако је постављен на вредност 1 означава да се не дозвољава деоба пакета. То је команда рутеру да не дели пакет пошто одредиште није у могућности да делове спаја натраг у једну целину. Предност пакета са постављеним DF битом је та што пошиљалац зна да ће он стићи на одредиште као једна целина. Недостатак је што се може десити да се пакет одбаци уколико на својој путањи премаши максималну величину пакета коју нека од успутних мрежа може да подржави. Да се ово не би десило било би добро користити изворишно рутирање[2] да би се заобишле мреже које прихватају само мале пакете;
    - ✓ ако је постављен на вредност 0 дозвољава се деоба.
  - Трећи бит означен са MF[3] :
    - ✓ постављен на вредност 0 указује да је ово последњи део у пакету,
    - ✓ постављен на вредност 1 указује да постоји још делова.
- Одступање делова[4] (дужине 13 бита) користи се као испомоћ у повезивању делова у оригинални пакет.
- Време трајања TTL[5] (дужине 8 бита) указује колико дуго (у секундама) је дозвољено пакету да буде „путује" по мрежи. Теоретски сваки рутер који обрађује пакет требало би да од вредности овог поља одузме време обраде пакета. практично сваки рутер обради пакет за мање од 1 секунде. Зато рутери само умањују за један вредности овог поља. Тако време трајања TTL постаје мера броја прелазака (скокова)[6] а не мера времена. Када вреднст TTL поља достигну вреднсот нула онда се претпоставља да се пакет креће у затвореној петљи и зато се одбацује. Иницијалну вредност треба да поставе протоколи виших слојева који и формирају пакет и иницирају његов пренос.

---

[1] *Don't Fragment*
[2] *Source routing* - унапред, на предајној страни (изворište) одреди путању (руту).
[3] *More Fragments* - још фрагмената
[4] *Fragment Offset*
[5] *Time To Live*
[6] *Hop metric*

- Ознака протокола[1] (дужине 8 бита) указује ком протоколу вишег слоја треба податке из овог пакета испоручити. Неке од вредности су дате у табели 4.2.

Децимална вредност поља	Протокол
0	Резервисано
1	Управљачке поруке на Интернету ICMP
2	Протокол за управљање IGMP
3	Протокол између мрежних пролаза GGP (*Gateway to Gateway Protocol*)
4	IP (IP укалуплъивање)
5	Ток (*Stream)*
6	TCP
8	Спољашњи протоколи за рутирање EGP (*Exterior Routing Protocol*)
9	Протокол за рутирање унутар приватних мрежа PIRP (*Private Interior Gateway Protocol*)
17	UDP
41	IPv6
50	Сигурносни протокол за IPv6 ESP (*Encap Security Payload for IPv6*)
41	IPv6
51	Заглавље за аутентификацију за IPv6 AH (*Authentication Header for IPv6*)
89	Протокол за рутирање OSFP (*Open Shortest Path First)*

Табела 4.1. Листа додељених бројева на Интернету[2]

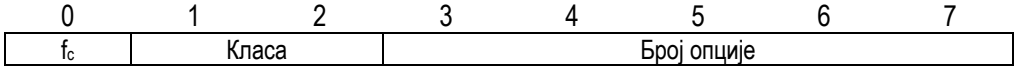
- Контролна сума заглавља (дужине 16 бита) односи се само на информације које су смештене у заглавље. Уочимо да се она мора израчунавати на свакој траси пошто се најмање једно поље мења (нпр.TTL). Пакет се одбацује уколико контролни збир није у реду;
- Изворишна адреса (дужине 32 бита) је IP адреса станице која шаље пакет;
- Одредишна адреса (дужине 32 бита) је IP адреса станице којој је упућен пакет;
- Опције (променљиве дужине) имају формат приказан на слици 4.25 или само 1 бајт (врста);

1 бајт	1 бајт	Дужине	2	бајта
Врста[3]	Дужина[1]	Опциони подаци[2]		

[1] *Protocol Number*
[2] Комплетна листа додељених бројева може се наћи у STD 2.
[3] *Type*

Слика 4.25. Формат поља опције

У оба случаја поље „врста” има структуру приказану на слици 4.26.



Слика 4.26. Структура поља „врста”

Где је:

- f_c[3] је поље које указује да ли ће се (ако је вредност поља 1) или се неће (ако је вредност поља 0) копирати поље опције приликом деобе (фрагментације) пакета,
- класа је поље које садржи 2-битни број без знака:
  - ✓ 0 - управљање,
  - ✓ 1 - резервисано,
  - ✓ 2 - откривање грешке и мерење,
  - ✓ 3 - резервисано.
- Описаћемо само неке од опција:
  - ✓ 2 је вредност поља „број опције" и односи се на сигурност. Класа је 0, f_c бит је постављен на вредност 0, поље „дужина" има вредност 11 и 8 бајтова за податке. Користи се за безбедносне информације које су потребне DoD[4];
  - ✓ 3 је вредност поља „број опције" и односи се на незаобилазно рутирање[5]. Класа је 0, f_c бит је постављен на вредност 1, и променљиве је дужине поље за податке. Извориште специфицира листу рутера које не треба заобићи;
  - ✓ 4 је вредност поља „број опције" и односи на означавање времена[6]. Класа је 2, f_c бит је постављен на вредност 0 и променљиве је дужине поље за податке. Дозвољава сваком рутеру да додаеје своју адресу и време када је рутиран;

---

[1] *Length*
[2] *Option Data*
[3] *Flag copy*
[4] *U.S. Department of Defense*
[5] *Loose source routing*
[6] *Timestamp*

- ✓ 7 је вредност поља „број опције" и односи на прављење записа о рути[1]. Класа је 0, f_c бит је постављен на вредност 0 и променљиве је дужине поље за податке. Дозвољава сваком рутеру да додаје своју адресу;
- ✓ 9 је вредност поља „број опције" и односи на стриктно рутирање изворишта[2]. Класа је 0, f_c бит је постављен на вредност 1 и променљиве је дужине поље за податке. Даје комплетан пут који треба пакет да следи;

° Дужина је поље које садржи број бајтова опције узимајући у обзир и поље „врста" и „дужина";

° Опционе податке сачињавају подаци који се односе на ту опцију;

- Додатак (променљиве дужине) је део који се додаје пакету уколико дужина пакета (датаграма) није целебројни умножак од 32. Састоји се од свих нула;
- Подаци (променљиве дужине) се прослеђују вишим слојевима. Максимална дужина пакета: поља података и заглавља је 65 535 бајтова;

4.2 ИНТЕРНЕТ ПРОТОКОЛ ВЕРЗИЈЕ 6

Броја рачунара који је повезан на Интернет расте изузетно великом брзином. У јулу 2005. год Интернет је користило скоро 350000000 станица[3]. Са IPv4 величином адресног поља могло би да се обезбеди 2^{32} различитих адреса што је десет пута више него број станица који тренутно користи Интернет. Разлог због којих IPv4 адресни простор постао неодговарајућа су следећи:

- Двослојна структура IP адреса (број мреже и број станице) је добро решење, али је и непотребно трошење адресног простора. Када се број мреже једном додели одређеној мрежи, све станица на тој мрежи аутоматски добијају свој број (адресу). Адресни простор те мреже може бити неискоришћен, али ако се посматра ефекат на IP адресни простор онда када се искористи број мреже искоришћене су и све адресе за станица у оквиру те мреже;
- Адресни простор верзије IPv4 подељен је у мреже класа A, B и C различитих величина. Потребно је посебно анализирати адресни простор сваке од њих.
- Модел IP адресирања генерално захтева да јединствен мрежни број буде додељен свакој IP мрежи без обзира да ли јесте (или није) повезане на Интернет;

---

[1] *Record route*
[2] *Strict source routing*
[3] ISC *Internet Domain Survey,*Jan 2004, *http://www. isic.org*

- Број рачунарских мрежа расте великом брзином. Многе организације без икакве потребе користе уместо једне више локалних рачунарских мрежа. Бежичне мреже добијају на значају;
- Проширивање области у којима се користе TCP/IP протоколи као што су POS[1] терминали или кабловска телевизија доводе до повећања потражње за IP адресама;

Видимо је да су потребе за већим адресним простором довела до потребе за новом верзијом IP протокола. Протокол IP је дуго у употреби а у међувремену су се појавили нови захтеви везани за конфигурацију адреса, рутирања и механизмима подршке у управљању саобраћајем. Проблеме које IPv4 протокола није мога да разреши су и:

- У чворовима (рутерима) Интернета (нарочито у окосницам)а табеле рутирања су постале превелике да би се могле одржавати, без обзира на примену CIDR рутирања;
- Апликације у реалном времену (које се све више користе) захтевају да постоје класе сервиса и механизми приоритета. Код верзији IPv4 механизми који обезбеђују квалитет сервиса нису били јасно дефинисани и веома су се мало користили;

Организација IETF[2] је као одговор на постојеће пробелеме[3] 1992. год. започела процедуру за нову верзију Интернет протокола. До данас су објављени бројни документи међу којима су најважнији: RFC 1752[4], RFC 2460[5], RFC 2373[6]

Разлике између верзија IPv6 и верзије IPv4 су следеће:

- Дужина IPv6 је повећа на 40 бајтова (са 20 бајтова IPv4) и садржи две адресе од по 16 бајтова (изворишта и одредишта). Заглавље IPv4 (слика 4.22) сарджи две адресе од по 4 бајта иза кога је 12 бајтова управљачих информација и обично опциони подаци. Проширење адресног простора је за фактор 2^{96}. Израчунато је да ово омогућава $6x10^{23}$ јединствених адреса по квадратном метру површине Земље[7]. Како год се адресе додељују, овај адресни простор је довољно велики. Предвиђања су да ће бити довољан за следећих 30 година;
- Намера пројектаната верзије IPv6 је да се смањи време обраде пакета у рутерима смањењивањем броја управљачких информација и измештање опционих података из заглавља. Опције IPv6 смештене су у посебна заглавље

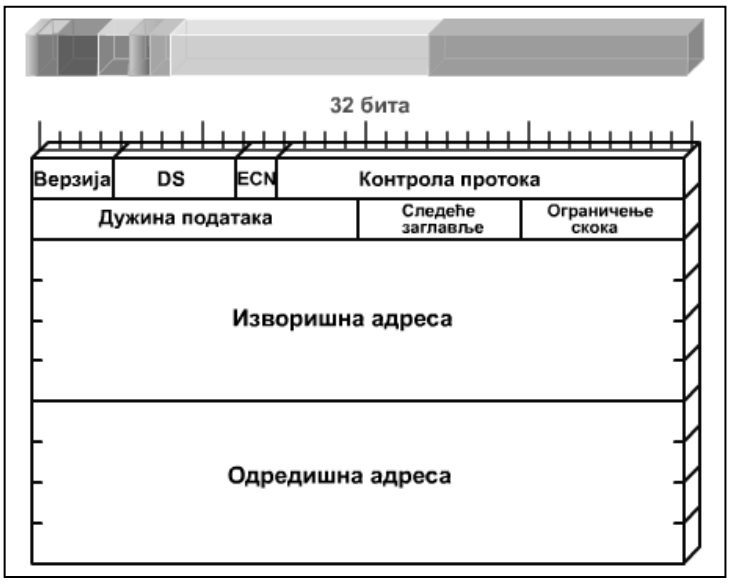
[1] *Point Of Sale*
[2] *Internet Engineering Task Force*
[3] *IP Address Exhaustion Problem*
[4] Сматра се прекретницом у дефинисању новог протокола означеног са IPng. (*Recommendation for the IP Next Generation*).
[5] Односи се на свеобухватну спецификацију протокола IPv6.
[6] Односи се на структуру адресирања IPv6.
[7] [Hind95]

која се постављено између заглавља IPv6 и заглавља транспортног слоја. Највећи број додатних заглавља рутери на путу који пакета прелази се не анализирају;

- Повећана флексибилност адресирања: IPv6 укључује и концепт адресе један ка једном од групе (еникаст) Пакет се испоручује само једним путем. Перформансе рутирања један ка групи (мултикаст) су унапређене тако што је додат опсег поља за мултикаст адресе;
- Подршка у додели ресурса: IPv6 омогућава означавање пакета за различите брзине проток ако то пошиљалац тражи. На располагању су и додатне услуге у саобраћају када се преносе видео сигнал[1].

Формат заглавља протокола IPv6

Заглавље протокола IPv6 је поједностављено у односу на заглавље претходника IPv4. Формат заглавља IPv6 предствљен је на слици 4.28.



Слика 4.28. Формат IPv6 заглавља

Састоји се од следећих поља:

- Верзија (дужине 4 бита): верзија Интернет протокола, вредност је 6;
- DS/ECN[1] (дужине 8 бита): првих шест бита поља односни се на поље DS[2], а остала 2 бита резервисана су за обавештења о загушењу ECN[3];

[1] *Real-time video*

- Ознака тока (дужине 20 бита): може користити станица да обележи оне пакете са којима рутери треба да поступају на посебан начин у оквиру мреже;
- Дужина података (дужине 16 бита): дужина остатка IPv6 пакета који прати заглавље, у октетима. Другим речима, ово је комплетна дужина свих заглавља и јединице података транспортног слоја TPDU[4];
- Следеће заглавље (дужине 8 бита): идентификује тип заглавља које прати IPv6 заглавље. Ово може бити неко од IPv6 додатних заглавља или заглавље вишег слоја, као што су заглавља TCP или UDP протокола. Поред оних вредности које су идентичне са верзијом IPv4 ово поље може да има следеће вредности:
  - 41- IPv6 заглавље,
  - 45 - међудоменски протокол за рутирање[5],
  - 58 -ICMP пакет протокола IPv6,
  - 46 - протокол за резервисање ресурса[6].

  Вредности за додатна заглавља су следеће:
  - 0 - заглавље опције корак по корак ,
  - 43 - заглавље IPv6 рутирања ,
  - 44 - заглавље IPv6 делова,
  - 50 - заглавље сигурности податка,
  - 51 - заглавље о IPv6 аутентификацију,
  - 59- нема следећег заглавља[7],
  - 60 - заглавље одредишних опција.
- Ограничење скока (дужине 8 бита): колико је још скокова за овај пакет дозвољено. Ограничење скокова поставља изворište одговарајућу вредност.

---

[1] У првим докумнетима ово поље је означавано као "класе саобраћаја" (*Traffic Class*). Његова употреба је резервисано за почетне чворова и/или прослеђујуће рутере да би се направила разлика између различитих класа приоритета IPv6 пакета.
[2] *Differentiated services*
[3] *Explicit congestion notification*
[4] TPDU (*Transport Protocol Data Unit*)
[5] *Interdomain Routing Protocol*
[6] *Resource Reservation Protocol*
[7] *No Next Header*

Декрементира се у сваком чвору који прослеђује пакет. Када вредност овог поља постане нула пакет се одбацује[1];

- Изворишна адреса (дужине 128 бита): адреса оног који пакета формира;
- Одредишна адреса (дужине 128 бита): адреса оног који пакета прима. Ово не мора да буде крајња одредишна адреса ако је присутно заглавље рутирања;
- Контролна сума: протокол IPv6 не израчунава контролну суму зато што траснпортни протоколи[2] то већ раде и зато што постоји опција за аутентификацију која може да обезбеди интегритет података.

Иако је IPv6 заглавље дуже него обавезни део IPv4 заглавље (40 према 20 бајтова) број поља је мањи (8 према 12). То значи да ће рутери имати мање посла око обраде заглавља чиме је процес рутирање значајно убрзан.

Структура IPv6

Пакет IPv6 има општу структуру приказану сликом 4.26. Сваки IPv6 пакет започиње са основним заглављем. У многим случајевима ово ће бити и једино заглавље које је потребно да би се пакет испоручио одредишту. Некада је потребно пренети и још неке податке[3] одредишту или рутерима на путу до одредишта. Додатна заглавља се користе у ту сврху а постављају се одмах иза IPv6 заглавља. Свако додатно заглавље (осим оног које носи број 59) има своје поље[4] које означава који тип (врста) заглавља следи иза њега. Оваква структура омогућава уланчавање различитих додатних заглавља (проширења).

Слика 4.26 Форма пакета IPv6

Дефинисана су следећа додатна заглавља (проширења):

- заглавље опције корак по корак[5]: дефинише специјалне опције које захтевају обраду корак по корак,
- заглавље IPv6 рутирања[1]: обезбеђује проширење рутирања, слично изворишном рутирању у IPv4,

---

[1] Ово је поједностављен поступак у односу на поступак који би требало да се урадити са TTL пољем код IPv4. Рад са временским интервалима код верзије IPv4 није донео никакву предност. Уствари рутери са IPv4 протоколом третирали су TTL поље само као ограничење у броју скокова.

[2] Код израчунавања контролне сумеTCP и UDP користе псеудозаглавље у коме је садржана и IP адреса.

[3] Код верзије IPv4 тај задатак је обављало поље опција.

[4] *Next Header Field*

[5] *Hop-by-Hop Options header*

- заглавље IPv6 делова[2] (фрагмента) пакета: садржи информације о дељењу и поновном повезивању пакета,
- заглавље о IPv6 аутентификацију[3]: обезбеђује интегритет и аутентичност сваког пакета,
- заглавље о сигурности података[4]: обезбеђује заштиту пакета,
- заглавље одредишних опција[5]: садржи опционе информације које ће одредишни чвор анализирати.

У случају када се користи више додатних заглавља IPv6 препоручује да се заглавља појављују следећим редоследом:

1. заглавље IPv6 (обавезно мора да буде прво),
2. заглавље опције корак по корак,
3. заглавље одредишних опција - за опције које ће бити обрађене од стране првог одредишта које се појављује у пољу IPv6. Одредишне адресе и накнадна одредишта која су набројана у заглављу рутирања,
4. заглавље IPv6 рутирања,
5. заглавље IPv6 делова,
6. заглавље аутентификације,
7. заглавље сигурности података,
8. заглавље одредишних опција - за опције које ће бити обрађене само у крајњем одредишту пакета.

---

[1] *Routing header*
[2] *Fragment header*
[3] *Authentication header*
[4] *Encapsulating Security Payload header*
[5] *Destination Option header*

Слика 4.27 IPv6 пакет са продуженим заглављима

Слика 4.27 приказује пример једног IPv6 пакета који укључује део сваког заглавља осим оних која су повезана са сигурношћу. IPv6 заглавље и свако продужено заглавље садрже поље „следеће заглавље". Ово поље идентификује тип следећег заглавља. Ако је следеће заглавље једно од продужених заглавља, онда ово поље садржи идентификатор типа тог заглавља. У противном, ово поље садржи идентификатор протокола вишег слоја који користи IPv6 (обично протокол транспортног слоја). На слици 4.27 као протокол вишег слоја је постављен TCP протокол. Подаци (виших слојева) које носи IPv6 пакет садрже заглавља TCP протокола и податке апликације. Пример уланчавања додатних заглавља дат је на слици 4.28.

Слика 4.28. Пример уланчавања

Ознака тока[1]

IPv6 стандард дефинише ток као низ пакета који се шаљу од одређеног изворишта ка одређеном одредишту[2] и за које изворište захтева од рутера да на посебан начин поступа. Проток је јединствено дефинисан тако што се комбинују изворишна и одредишна адреса са ознаком тока (20-битни број различит од нуле). Изворište додељује исту "ознаку тока" свим пакетима који су део једног тока.

Посматрано са стране изворишта ток је низ пакета које је генерисала апликација изворишта а који захтева исти транспортни сервис. Ток може да обухвати једну или више TCP веза. Једна апликација може да генерише један или више токова. Пример два тока је мултимедијални састанак, коме је додељен један ток за звук а други за слику. Сваки од

[1] *Flow label*

[2] Уникаст (*unicast*) или мултикаст (*multicast*).

токова има различите захтеве код преноса података, кашњења или одступање у кашњењу.

Ако се погледа из угла рутера, ток је низ пакета који има исте атрибуте који утичу на то како ће се рутер понашати према пакетима тог тока. Ту спада: одабрана путања, додељени ресурси, критеријуми за одбацивања, алгоритам заштите итд. Рутер може да се понаша другачије према пакетима који нису из истог тока. На пример: доделом различитог меморијског простора (бафера), придруживањем различитог приоритета код прослеђивања пакета или различитим захтевом за квалитетом сервиса који се од мреже очекује .

Не постоји посебно значење било које ознаке тока. Уместо тога, да би се обезбедио специјални поступак ток се мора обележити на неки други начин. На пример, изворište може да затражи од рутера специјални поступак везан за обраду времена помоћу протокола за управљање или времена преноса тако што ће поставити одговарајуће информације у једно од додатних заглавља пакета, као што је нпр. заглавље опције "корак по корак". Примери специјалног поступка су и захтеви за нестандардним квалитетом сервиса или неким од сервиса у реалном времену.

У начелу, сви кориснички захтеви за одређеним током могу бити дефинисани у додатним заглављима и укључени у сваки од пакета. Ако се жели оставити концепт тока потпуно отвореним за мноштво захтева, резултат би могао бити превелико заглавље пакета. Друга могућност, која је усвојена код протокол IPv6, је тзв. ознака тока код које су захтеви тока дефинисани на самом почетку а јединствена ознака тока додељена је сваком току. У овом случају рутер мора да чува информације о захтевима сваког тока.

Следећа правила важе за ознаке тока:

1. Станица или рутер који не подржавају поље "ознака тока" мора да када прави пакет постави поље на нулту вредност, када преусмерава пакет не мења његову вредност а игнорише га на пријему;
2. Сви пакети који су направљени у истом изворишту морају имати исту: вредност поља "ознака протока", одредишну адресу, изворишну адресу, садржај заглавља опције "корак по корак" (ако постоји) и садржај заглавља "рутирање" (ако постоји). Рутер може да обрађује пакет тако што ће једноставно погледати ознаку тока у табели без испитивања осталих заглавља;
3. Изворište додељује току "ознаку тока". Нова ознака тока мора бити изабрана као случајни број из опсега 1 до 2^{20} -1. Услов је да изворište не може да додели ознаку новом току ако се она користи за ток који је још активан. Вредност нула указује да се ни једна од ознака тока не користи;

Адресирање протокола IPv6

Код протокола IPv6 адресе су дужине 128 бита. Адресе се додељују сваком од интерфејсу чвора[1] а не самом чвору. Сваки интерфејс може да има више јединствених уникаст адреса. Било која од уникаст адреса додељена интерфејсу чвора може се искористити да јединствено идентификује тај чвор.

Комбинација адреса велике дужине и више адреса додељених интерфејсу омогућава ефикасније рутирање у поређењу са верзијом IPv4. Код IPv4 протокола, адреса није направљена тако да потпомаже процес рутирањуа. Због тога рутер је морао да води рачуна о великим табелама путања (рутирања). Дуже Интернет адресе дозвољавају повезивање већег броја адреса према: хијерархијама рачунарских мрежа, добављачима Интернет услуга преко којих се приступа Интернету, географској лоакацији предузећу итд. Такав број би требало да учини да табеле рутирања буду мање, а преглед табела бржи. Претплатнику је омогућено да приступа Интернету преко више добављача Интернет услуга а да за то користи исти интерфејс.

IPv6 дозвољава три типа адреса:

- Уникаст адреса је идентификатор једног интерфејса. Пакет послат уникаст адресом испоручује се интерфејсу који је одређен том адресом;
- Еникаст адреса је идентификатор за више интерфејса који припадају различитим чворовима. Пакет послат еникаст адресом испоручује се неком од интерфејса који је одређен том адресом (најближем, гледајући по протоколу за рутирање);
- Мултикаст адреса је идентификатор за групу интерфејса који припадају различитим чворовима. Пакет послат мултикаст адресом испоручује се свим интерфејсима који су одређени том адресом;

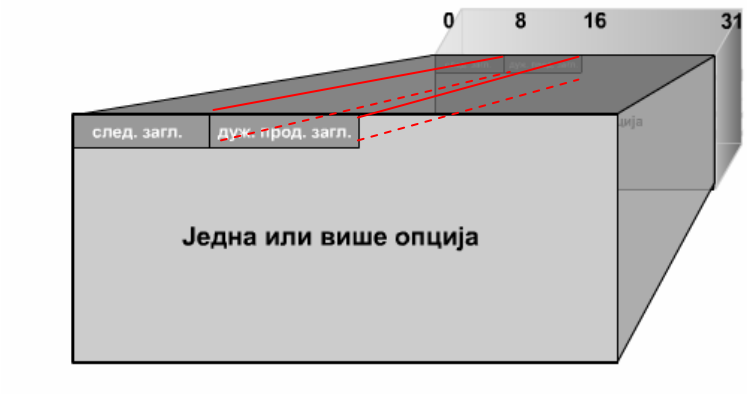
Заглавље опција корак по корак

Заглавље опција корак по корак носи опционе информације које, уколико је присутно, мора да анализира сваки рутер који се налази на путањи тог пакета. Састоји се (слика 4.29) од следећих поља:

- Следеће заглавље[2] (дужине 8 бита): одређује тип заглавља које се налази иза овог заглавља;
- Дужина заглавља[3] (дужине 8 бита) означава дужину тог заглавља исказаног у јединицама од по 64 бита, не укључујући прва 64 бита;
- Опције[4]: поље променљиве дужине које се састоји од једног или више поља која дефинишу опцију. Свака од дефиниција састављена је од три подпоља:

[1] Код протокола IPv6 термин чвор се односи и на рутер и на крајњу станицу. Односно на сваки уређај код кога је имплементиран IPv6 протокол.
[2] *Next header*
[3] *Header extension length*
[4] *Options*

- тип опције (дужине 8 бита) које одређује опцију,
- дужина опције (дужине 8 бита) које одређује дужину поља опције података у бајтовим и
- подаци опције (променљиве дужине) која представља спецификација опције.



Слика 4.29 Опције корак по корак

Да би се описала нека одређена опција у пракси се из поља "тип опције" користи пет битова најмање тежине. Два бита највеће тежине указују на активност коју чвора треба да одради уколико не препознаје тип опција:

- 00 – прескочи ову опцију и настави са обрадом заглавља,
- 01 – одбаци пакет,
- 10 – одбаци пакет и пошаљи ICMP поруку о проблему са параметрима[1] изворишној адреси пакета, што указује на немогућност препознавања типа опција,
- 11 – одбаци пакет само под условом да одредишна адреса није мултикаст адреса, пошаљи ICMP поруку о проблему са параметрима изворишној адреси пакета, указујући на непрепознатљив тип опција.

Трећи бит највеће тежине одређује да ли се на путу од изворишта до одредишта поље "опција података" не може (вредност 0) или може мењати (вредност 1). Подаци који се могу мењати морају бити искључени код израчунавања аутентификације.

Конвенције које се користе код поља тип опција односи се и на заглавље "одредишне опције".

За сада су дефинисане четири опције корак по корак :

[1] *ICMP Parameter Problem*

- Додатак 1[1]: користи се да би се уметнуо један бајт (додатак) у делу заглавља за опције.
- Додатак N: користи се да би се уметнуло N ($N \geq 2$) бајтова (додатка) у делу заглавља за опције.
- Велики део за податке[2]: користи се да би се послао IPv6 пакет са корисним подацима већим од 65 535 бајтова. Поље "опција података" је дужине 32 бита и указује на дужину пакета изражену у бајтовима изузимајући IPv6 заглавља. За ове пакете, вредност поље "дужина података" у IPv6 заглављу мора бити постављена на нулу и не сме бити присутно заглавље фрагмената. Са овом опцијом протокол IPv6 подржава величину пакета до 4 милиона бајтова. На тај начин омогућен је пренос великих видео пакета и омогућава IPv6 протоколу да на најбољи могући начин искористи капацитет трансмисионих медијума.
- Упозорење рутера[3]: информише рутер да је садржај пакета користан за рутер и да треба да поведе води рачуна о управљачким подацима који су њему. Изостанак ове опције у IPv6 пакету је знак рутеру да пакет не садржи информације које су њему од значаја. Због тога рутер ове пакете може проследити даље без рашчлањавања и анализе појединих делова. Станице које генерише IPv6 пакете треба ову опцију да користи само под одређеним условима. Опција обезбеђује ефикасну помоћ протоколима као што су протокол RSVP[4] који генерише пакете које рутери у циљу управљања саобраћајем треба да рашчлањују и анализирају.

Заглавље дела (фрагмента) пакета

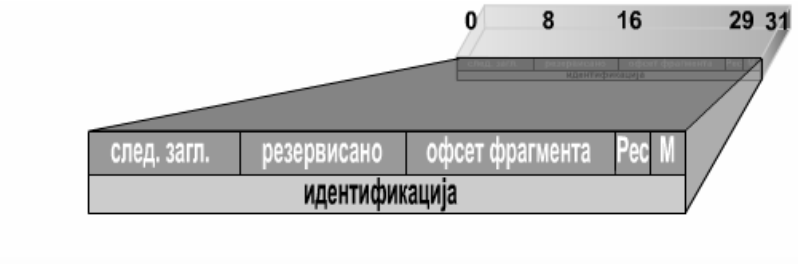
Код протокол IPv6 дозвољено је да деобу уради само изворишне станице а не и рутери преко који пакет прелази на свом путу до одредишта. Да би се искористила комплетне предности међуумрежавања изворишна тачка мора применити алгоритам за откривање пута који јој омогућава да сазна која је то најмања максимална јединица за пренос MTU[5] коју рачунарске мрежа на путу пакета од изворишта до одредишта подржавају. Када сазна величину јединице MTU[6] изворишна станица може, узимајући тај податак у обзир, пакет да подели. Уколико изворište нема податак о јединице MTU онда он мора да величину пакета ограничи на 1280 бајтова, колика је минимална јединица MTU коју свака рачунарска мрежамора мора да подржи.

Заглавље дела пакета састоји се од следећих поља (слика 4.30):

---

[1] *Pad*
[2] *Jumbo payload*
[3] *Router alert*
[4] *Resource Reservation*, дефинисан документом RFC 2205
[5] *Maximum Transmission Unit*
[6] Објашњено је у документу RFC 1191 (*Path MTU Discovery*)

- Следеће заглавље[1] (дужине 8 бита): одређује тип заглавља које се налази иза овог заглавља;
- Резервисано[2] (дужине 8 бита): за будућу употребу;
- Одступање дела пакета[3] (дужине 13 бита): указује где је у оригиналном пакету место корисних података[4] фрагмента. Мери се у јединицама од по 64 бита. Ово значи да део пакета (осим последњег) мора да садржи поље података које је умножак од 64 бита.
- Резервисано[5] (дужине 2 бита): резервисано за будућу уптребу;
- M[6] ознака (дужине 1 бит): када је M = 1 значи да има још делова а када је M = 0 значи да је то последњи део оригиналног пакета.
- Идентификација[7] (дужине 32 бита): служи за јединствену идентификацију оригиналног пакета. Идентификатор мора бити јединствен за изворишну и одредишну адресу пакета за време док је он на Инетрнету. Сви делови пакета са истим идентификатором, изворишном и одредишном адресом поново се повезују како би се добио оригинални пакет.



Слика 4.30 Заглавље дела (фрагмената) пакета

Начин деобе (фрагментације) је исти као што је описано у делу о IPv4.

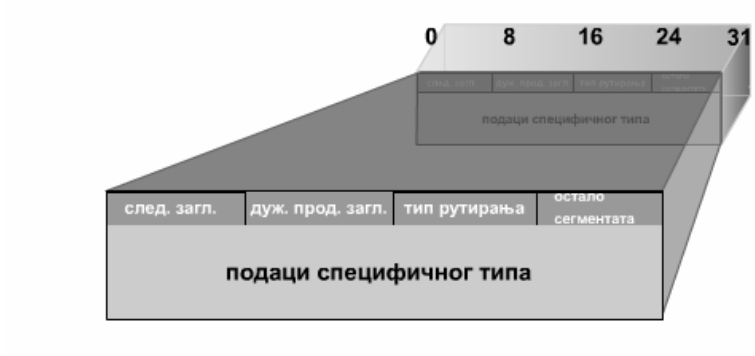
Заглавље дела за рутирања

Заглавље дела за рутирања садржи листу једног или више чворова преко којих пакет треба да пређе на свом на путу до одредишта. Заглавље дела за рутирања почиње са 32-битним делом који се састоји од четири 8-битних поља, иза којих следи поље са

[1] *Next Header*
[2] *Reserved*
[3] *Fragment Offset*
[4] *Payload*
[5] *Res*
[6] *M Flag*
[7] *Identification*

подацима за рутирања који су специфични за одређени тип рутирања (слика 4.31). Четири 8-битна поља су следећа:

- Следеће заглавље[1]: одређује тип заглавља које се налази иза овог заглавља;
- Дужина заглавља[2]: означава дужину тог заглавља исказаног у јединицама од по 64 бита, не укључујући прва 64 бита;
- Тип рутирања[3]: идентификује одређену варијанту заглавља рутирања. Ако рутер не препозна вредност типа рутирања, мора да одбаци пакет;
- Преостали сегменти[4]: број преосталих сегмената рутирања. Значи указује на број преосталих чворова (од оних који су експлицитно наведени) преко којих пакет још треба да пређе на путу до одредишта.



Слика 4.31 Заглавље рутирања

Једино специфично заглавље рутирања које је дефинисано докумнтом RFC 2460 је заглавље рутирања нултог типа (слика 4.32). Када се користи Тип 0 заглавља рутирања, изворишна тачка не поставља крајњу одредишну адресу у IPv6 заглавље. Уместо тога, адреса је она која је последња у листи адреса у заглављу рутирања, а IPv6 заглавље садржи адресу првог жељеног рутера на траси. Заглавље рутирања неће бити испитивано док пакет не дође до тачке коју је наведена од стране IPv6 заглавља. У тачки, садржаји заглавља рутирања и IPv6 заглавља се ажурирају и пакет се прослеђује даље. Ажурирање се састоји од постављања следеће адресе коју треба посетити у IPv6 заглавље и декрементирању поља Остало сегмената у заглављу рутирања.

Слика 4.32 заглавље рутирања нултог типа.

Заглавље одредишних опција

---

[1] *Next Header*
[2] *Heder Extension Length*
[3] *Routing Type*
[4] *Segments Left*

Заглавље одредишних опција, носи опционе информације које, уколико је заглавље присутно, испитује само одредиште пакета. Формат овог заглавља је исти као и формат заглавља опција корак по корак (слика 4.29).

5. АЛГОРИТМИ ЗА РУТИРАЊЕ

Алгоритам за рутирање је софтверски део мрежног слоја одговоран за одлуку на коју од излазних линија пристигли пакет ће бити послат. Када подмрежа (без успоставе везе) користи датаграме одлука се доноси за сваки пристигли пакет података пошто се оптимална путања можда у међувремену променила. Када подмрежа (са успоставом везе) користи виртуелно коло одлуке о рутирању се доносе при успостављању везе и пакети се шаљу том утврђеном путањом.

Корисно је направити разлику између преусмеравања (рутирања) и прослеђивања. Рутер има два процеса у себи:

- процес који анализира доспели пакет и проналази у табели рутирања излазну линију на коју ће пакет бити послат. Овај процес назива се прослеђивање[1].
- процес који је одговоран је за попуњавање и иновирање табела рутирања. Овај процес се назива преусмеравња (рутирање). Кључну улогу у овом процесу заузима алгоритам за рутирање.

Пре покушаја изналажења решења задатак алгоритма за рутирање је да донесе одлуку шта је то што треба оптимизовати. На пример: смањење средњег кашњења пакета је добар избор, али је такође добар избор и повећање укупног саобраћаја (протока) кроз мрежу. Ова два циља су у међусобној супротности пошто сваки систем са великим саобраћаје повлачи и дуже време чекања у редовима за обраду пакета. Као компромис покушало се са смањењем броја рутера[2], јер се тако смањује кашњење али и искоришћеност пропусног опсега што даље погоршава проток.

Постоје два начина формирања табеле за рутирање, па у складу с тим и алгоритми за рутирање се могу поделити у две главне групе: неприлагодљиви (статичко рутирање) и прилагодљиви (динамичко рутирање).

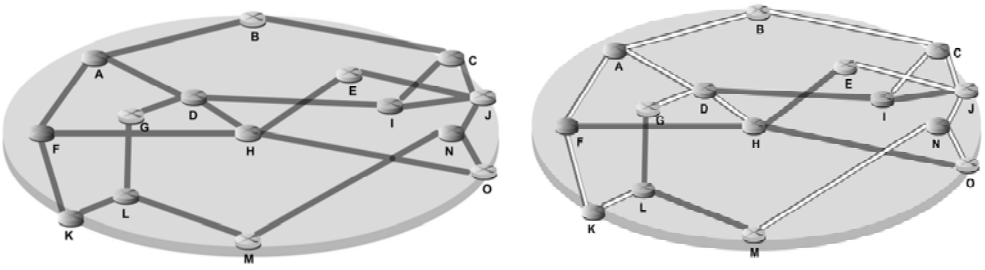
Пре анлизе конкретних алгоритама, потребно је дефинисати принцип изналажења оптималне путање који не зави од топологије и саобраћаја мреже. У литератури се среће под називом принцип оптималности и може се описати на следећи исказом: "Ако се рутер J налази на оптималној путањи од рутера I до рутера K, онда је оптимална путања од J до K део тог пута"[3].

---

[1] *Forwarding*

[2] Уобичајен термин је да се пролазак пакета кроз рутер назаива скок (енг. *hop*).

[3] Доказ се може наћи у литератури[Tanenbaum].



а) б)

Слика 5.1. а) Подмрежа, б) понирућe стабло за рутер B

Као директна последица принципа оптималности, можемо видети да скуп оптималних путања из свих извора ка задатом одредишту сачињава стабло с кореном у одредишту. Такво стабло зове се понирућe стабло[1]. На слици 5.2 приказана је подмрежа и *понирућe* стабло за одредиште B, где је као мера растојања узет број скокова. Циљ свих алгоритама за рутирање је да открију и користе *понирућа* стабла за све рутере.

Понирућe стабло не садржи петље, па се сваки пакет преносе са коначним бројем скокова. У пракси, то није тако једноставно. Везе и рутери могу да откажу и наставе касније са радом тако да различити рутери могу да имају различите представе о тренутној топологији мреже. У сваком случају принцип оптималности и понирућe стабло представљају стандардну подлогу за процену других алгоритама за рутирање.

Рутирање најкраћим путем[2]

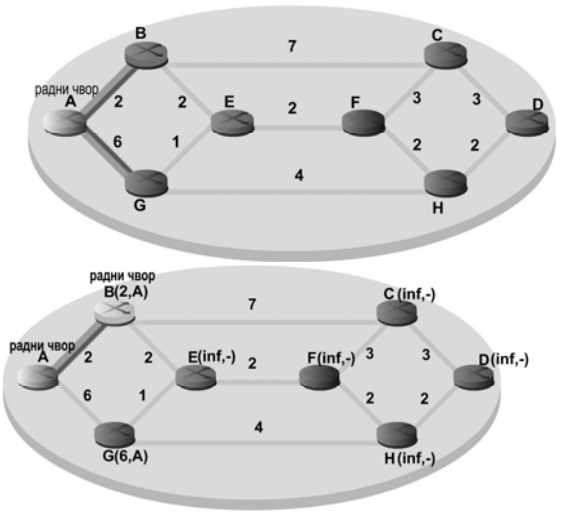
Метод који се назива рутирање најкраћим путем је широко распрострањен. Идеја се састоји у томе да се направи граф подмреже; сваки чвор у графу представља рутер, а свака линија у графу комуникациону линију[3]. За одабир путање између два задата рутера, алгоритам проналази најкраћу путању на графу.

Концепт најкраће путање заслужује објашњење. Један начин за мерење дужине пута је број скокова. Користећи овај начин мерења раздаљине[4], путеви ABC и ABE на слици 5.2 су једнаке дужине. Други начин мерења раздаљине може бити географска удаљеност. У том случају ABC је очигледно знатно краћа од ABE.

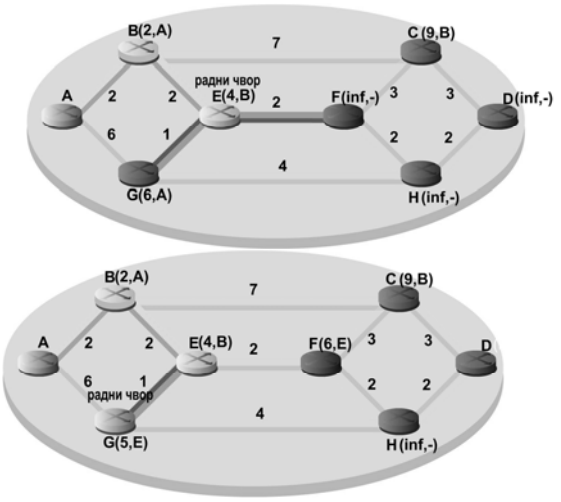
Поред поменутих користе се и други начин мерења удаљености: нпр. свака веза се може обележити средњим временом чекања или средњим кашњењем у преносу, а

[1] *Sink tree*
[2] *Shortest Path Routing*
[3] Линка, везе или линије.
[4] Користи се термин метрика.

вредности би се проверавале сваког сата[1]. У овом случају најкраћи пут је онај који је најбржи а не онај који је географски најкраћи .

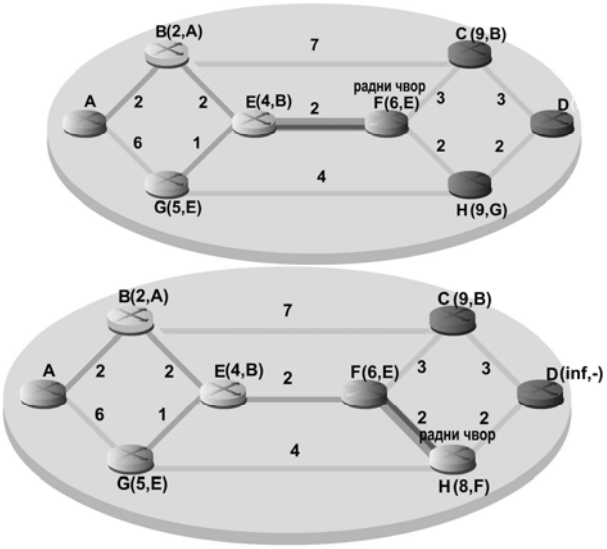


(а) (б)



(в) (г)

[1] Код оваквих мерења користи се унапред дефинисана величина пакета (тест пакет)



(д) (ђ)

Слика 5.2 Првих 6 корака у рачунању најкраћег пута од A до D

У општем случају, ознаке на линијама графа могу бити израчунате као функција удаљености, пропусног опсега, просечног саобраћаја, трошкова комуникације, средњег времена чекања у реду, средњег кашњења и других чинилаца. Алгоритам рачуна најкраћи пут одабирајући један критеријум или пак комбинацију више њих.

Познато је неколико алгоритама за рачунање најкраћег пута између два чвора. У следећим поглављима анализираћемо неке од њих: алгоритам Дикстра[1] и алгоритам преплављивања[2].

Алгоритам Дајкстра

Сваки чвор обележен је (на слици 5.3 ознаке у заградама):

- удаљеношћу од изворног чвора дуж најбоље познате путање,
- ознаком чвора из кога се врши испитивање.

У почетку, путање су непознате, па су сви чворови обележени са ознаком бесконачно "∞". Док алгоритам ради и проналази боље путање ознаке се мењају. Ознаке могу бити

[1] *Dijkstra* (1959)
[2] *Flooding*

променљиве и сталне. У самом почетку све су ознаке променљиве. Кад се установи да одређена ознака представља најкраћу путању од извора до одговарајућег чвора, она постаје стална ознака.

За илустрацију функционисања овог алгоритма служи слика обележеног неусмереног графа на слици 5.2, где ознаке на линијама представљају удаљености. Тражи се најбољи пут од A до D. За почетак, чвор A је обележен сталном ознаком. Самим тим, A постаје и радни чвор, што нам показује попуњени круг који озаначава чвора A. Затим се прегледа сваки од суседних чворова чвора A, један за другим. Они се означавају удаљеношћу до радног чвора. Чвор поново обележава чвором из кога је испитиван, тако да се касније може реконструисати коначна путања. Када се испитају сви суседни чворови чвору A наставља се испитивањем променљивих ознаке у читавом графу. Чвор са најмањом вредношћу добија сталну ознаку и постаје нови радни чвор. На слици 5.2 то је чвор B.

Сад се прегледају суседни чворови чвору B. Ако је збир вредности ознака чвора B и удаљености од B до чвора који се посматра мањи од ознаке тог чвора, постоји краћи пут, па се тај чвор означава новом вредношћу. Пошто се сви суседни чворови прегледају, цео граф се претражује у потрази за најмањом вредношћу променљиве ознаке. Изабрани чвор добија сталну ознаку и постаје радни за следећи круг. На слици 5.2 представљено је првих пет корака алгоритма.

Извешћемо доказ да алгоритам исправно ради. На слици 5.2 (в) чвор E постаје стални чвор. Ако претпоставимо да постоји краћи пут од ABE, рецимо AXYZE, постоје две могућности: или је чвор Z већ постао сталан, или није. Ако јесте, онда је E већ испитан (у кругу који је следио круг у ком је Z постао сталан), тако да путања AXYZE није измакла нашој пажњи и стога не може бити краћа путања.

Сада размотримо случај кад Z још увек има променљиву ознаку. Или је ознака чвора Z већа или једнака оној на E, у ком случају AXYZE није краћа путања од ABE, или је пак мања, а тад би чвор Z, а не E, први постао сталан. Чвор E би тад био испитиван из Z. Дакле, ABE је најкраћа путања, из чега следи да је алгоритам исправан.

Алгоритам преплављивања

Алгоритам преплављивања такође спада у групу статичких алгоритама. Принцип је следећи: сваки долазећи пакет прослеђује на све излазне линије осим на ону са које је дошао. Да би се заштитили од великог умножавања пакета (које у крајњем случају може бити неограничено) постављају се бројачи скокова у заглавље сваког пакета. који се Бројач се умањује своју вредност за један(декрементира се) при сваком проласку кроз рутер (тј.скоку). Пакет се одбацује кад вредност бројача постане нула. Најбоље би било да се вредност бројача постави на вредност једнакој дужини пута од изворишта до одредишта. Уколико пошиљалац не зна дужину пута, вредност бројача се може поставити на највећу вредност једнаку величини мреже.

Други начин за избегавање слања истог пакета два пута је вођење евиденције о оним пакетима који се прослеђују. Ово се може постићи тако да изворишни рутер додаје секвенцијски број сваком пакету, који прими од изворишног хоста. Сваки од рутера дуж путање треба да има листу већ послатих секвенцијских бројева изворишног рутера и уколико се долазећи пакет већ налази у листи, он неће бити поново послат. У циљу спречавања неограниченог раста листе, додаје се бројач *k*. Вредност овог бројача означава да су сви пакети са секвенцијским бројевима мањим и једнаким овој вредности већ послати. Такви пакети се одбацују. Најзад, пуна листа испод *k* није ни потребна, с обзиром да је *k* ефикасно представља.

Мало практичнија варијанта алгоритма преплављивања је алгоритам селективног преплављивања. Када користе овај алгоритам рутери не прослеђује пакете на све линије већ само на оне које на први поглед иду у правом смеру.

Алгоритам преплављивања је за највећи број апликација неприменљив. Користи се у неким случајевима као што су:

- војне апликацијама где постоји сталној опасности да велики број рутера престане с радом ,
- код дистрибуираних база података за истовремено ажурирање свих база података,
- у бежичним мрежама где су све поруке које шаље једна станица на располагању другим станицама унутар опсега радио сигнала. Практично се користи алгоритам преплављивања,
- употреба овог алгоритма као референце за процену других алгоритама.

Алгоритам преплављивања бира увек најкраћу путању пошто испитује сваку од могућих путања. Чињеница је да ниједан други алгоритам не може да обезбедити краће кашњење. Нарвно само ако се претпостави да се не узмима у обзир време премашења[1] које сам процес поплаве генерише.

Рутирање помоћу вектора удаљености[2]

У савремене рачунарским мрежам даје се предност динамичким над статичким[3] алгоритмима за рутирање јер статички алгоритми не узимају у обзир тренутну оптерећеност мреже. Два динамичка алгоритма која се највише користе и која ћемо анализирати у следећим поглављима су:

[1] *Overhead*
[2] *Distance Vector Routing*
[3] *Dijkstra*, *Flooding*

- рутирање на основу ветора удаљеност[1] и
- рутирање на основу стања везе[2].

Код алгоритама за рутирање на основу вектора удаљености сваки рутер одржава табелу (тј. векторе) чије записе чине удаљеност (метрика) најбоље познате путање ка одредишту и везе (линије) која се користи за ту путању. Табеле се ажурирају разменом информација са суседима. Метрика може бити број скокова, кашњење у милисекундама, укупни број пакета који чекају дуж путање исл.

Претпоставља се да рутер зна удаљеност до сваког суседа. Ако је метрика скок, удаљеност је један скок. Ако је метрика дужина реда, рутер просто прегледа сваки ред. И Ако је метрика кашњење рутер шаље специјалну врсту пакета[3] у које се на пријему уписује временска ознака[4] и враћа што је могуће пре натраг .

На пример: претпостављамо да је за метрику узето кашњење а да рутер зна кашњење до сваког суседа. Сваких T милисекунди сваки рутер шаље свим својим суседима и прима од њих листу са својом процена кашњења до сваког одредишта. Замислимо да је једна од ових табела управо стигла од суседа X, где X_i представља процену суседа X колико треба времена да се од њега стигне до рутера i. Ако рутер зна да кашњење до суседа X износи m секунди, такође зна да може да стигне до рутера i преко рутера X за $X_i + m$ секунди. Уколико се изврши израчунавање за сваког од суседа, рутер може да сазна која је од процена најбоља и искористи је за своју нову табелу рутирања.

Овај процес ажурирања илустрован је на слици 5.3. Са леве стране слике приказана је подмрежа. Прве четири колоне у десном делу ове слике приказују векторе кашњења, које је рутер J примио од својих суседа рутера A, I, H и K. Рутер A указује да је:

- 12 ms кашњење до рутера B,
- 25 ms кашњења до рутера C,
- 40 ms до рутера D, итд.

Претпоставимо да је рутер J измерио и проценио да је кашњење до његових суседа:

- 8 ms кашњење до рутера A,
- 10 ms кашњење до рутера I,
- 12 ms кашњење до рутера H и

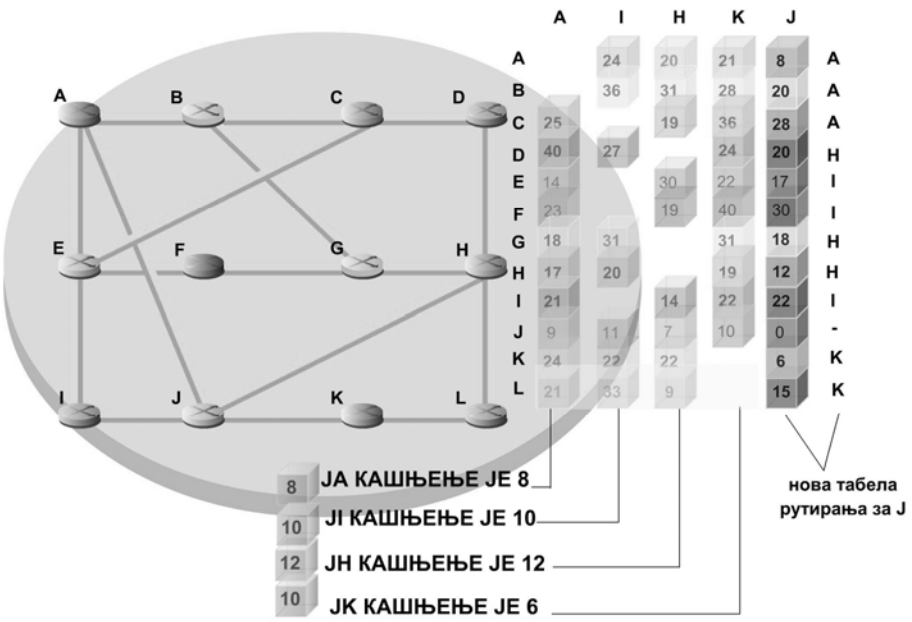
---

[1] D*istance vector routing*
[2] *Link state routing*
[3] ECHO
[4] *Timestamp*

- 16 ms кашњење до рутера K.

Како J рачуна своју нову путању до рутера G? Зна да стиже до рутера A за 8 ms, а рутер A тврди да може да стигне до рутера G за 18 ms. То значи да рутер J зна да може да рачуна на кашњење од 26 ms, ако пошаље пакете ка рутеру G преко рутера A. Слично, рутер J израчунава кашњење до рутера G преко рутера I, H и K и добија вредности 41 следећим редоследом: (31+10), 18 (6+12) и 37 (31+6). Најбоља од израчунатих вредности је 18 па се у табелу рутирања рутера J уноси кашњење од 18 ms и путања преко рутера H коју треба користити.

На исти начин врши се прорачун за сва остала одредишта. Нова табела рутирања за рутер J приказана је у последњој колони на слици 5.3.



Слика 5.3. Подмрежа, вектори из рутере A, I, H, K и нова табела рутирања за рутер J.

Проблем бројања до бесконачности

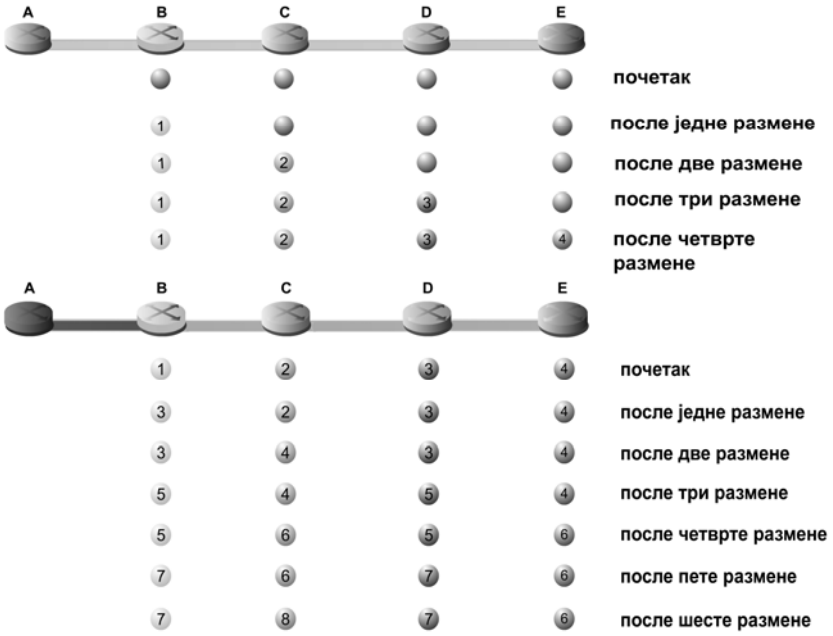
Алгоритам рутирање помоћу вектора удаљености теоријски је детаљно разрађен али се у самој реализацији јављају многи проблеми од којих ћемо поменути следеће:

- без обзира што се постижу добри резултати, често је тај процес спор (тзв. спора конвергенција),

- алгоритам брзо реагује на добре, а споро на лоше вести.

Посматрајмо рутер чија је најбоља путања до одредишта X веома дуга. Ако у следећој размени информација о рутирању сусед A извести о бољој метрици до X, рутер ће одмах почети да користи линију до рутера A и да преко њега шаље пакете ка X. Каже се да се добре вести брзо простиру (обрађују).

Да бисмо видели како се брзо шире добре вести, размотримо линеарну подмрежу од 5 чворова на слици 5.4, где се за метрику користи број скокова. Претпоставимо да је A у прекиду од самог почетка и да сви остали рутери то знају. Другим речима, свима је метрика до рутера A бесконачна.



Слика 5.4 Проблем бројања до бесконачности.

Кад се рутер A поново укључи у мрежу, други рутери ће то сазнати разменом вектора. Због једноставности претпоставићемо да се размена вектора врши синхронизовано у свим рутерима. Код прве размене вектора рутер B сазнаје да рутер A сада активан. Рутер B уписује у своју табелу рутирања да је рутер A на растојању од једног скока улево. Сви остали рутери и даље мисле да је рутер A у прекиду. У следећој размени вектора рутер C сазнаје да рутер B на растојању од једног скока од рутера A. Иновира своју табелу вредношћу растојањем до рутера A која је 2. У овој фази до рутера

D и E још увек нису допрле добре вести. Као што се види добре вест се прошире за један скока у току једне размени. У подмрежи чија најдужа путања сатављена од N скокова, кроз N размена сви рутери ће сазнати о поново успостављеним линијама и рутерима.

Сад размотримо ситуацију на слици 5.4 б), где су све линије и рутери иницијално исправни. Рутери B, C, D и E су удаљености од рутера A за 1, 2, 3 и 4 . Изненада рутер A престаје с радом или долази до прекида линије између A и B што је за рутеру B исто.

У првој размени пакета, B не успоставља комуникацију са A. Срећом, C се појављује са информацијом да има путању до A с метриком 2. Рутер B не зна да рутер C комуницира са рутером A преко њега самог. Као резултат, рутер B констаује да може да стигне до рутера A преко рутера C, са метриком 3. Рутери D и E не ажурирају своје записе о рутеру A при првој размени вектора.

У другој размени, C примећује да сваки од његових суседа тврди да има путању до A са метриком 3. Бира једну од њих случајним избором и ажурира своју удаљеност на 4. Резултати даљег тока размена вектора представљени су на слици 5.4 б).

Постепено, вредности се крећу ка "бесконачности"[1]. Овај проблем је познат као бројање до бесконачности. Да би се решио треба подесити "бесконачност" на вредност најдуже путање увећану за један. Суштина је у томе да када рутер X саопшти рутеру Y да има путању ка рутеру Z не постоји начин да рутер Y да утврди да ли се и он сам налази на тој путањи.

Протокол вектора удаљености

Релативно једноставан протокол интерног рутирања је протокол за размену информација о рутирању RIP[2]. Поред своје једноставности RIP протокол се показује ефикасним у мањим подмрежама и практично је највише коришћени протокол за рутирања.

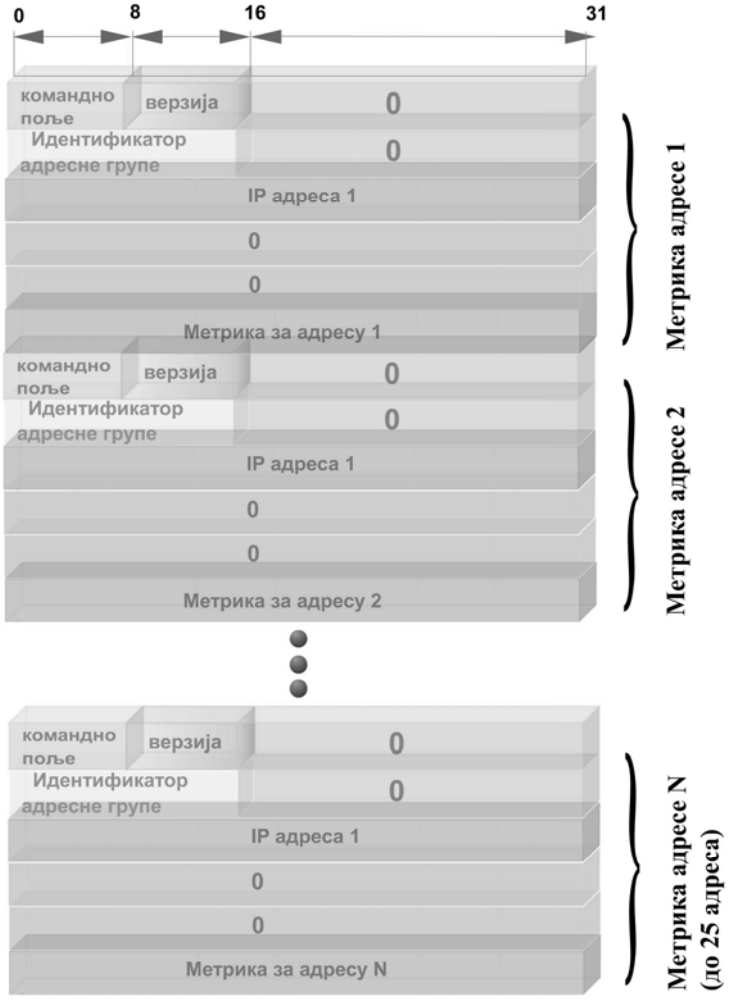
Кључна чињеница везана за протокол RIP је да он користи алгоритам рутирање заснован на вектору удаљености. Слика 5.5 показује формат RIP пакета. Сваки пакет укључује заглавље са следећим пољима:

- Управљачко - једно за захтев, друго за одговор. Измене стања рутирања се шаљу као одговору без обзира да ли су захтевани или не. Када чвор иницијализује RIP, он шаље RIP захтева упућен свим станицама (бродкаст). Сваки рутер прима захтев и тренутно шаље одговор;

[1] Као што се види појам бесконачности је симболичан: уколико се неби спречили процес увећавања метрике путање она би могла да порасте у недоглед.

[2] *Routing Information Protocol*

- Верзија - ово поље има вредност 1 за основни RIP протокол а вредност 2 за протокол RIP-2;



Слика 5.5 Формат RIP пакета

Иза заглавље налази се један или више блокова од којих сваки даје путању до одговарајућих одредишта мреже. Поменута поља имају следећи изглед:

- Адресна група - има вредност 2 за IP адресе;

- IP адресе - односи се на IP адресе код којих ни адреса мреже ни адреса станице не могу имати нулту вредност. На тај начин су поједине мреже јединствено дефинисане;
- Метрика - удаљености рутера од препознате мреже. Обично се употребљава вредност једног линка, тако да се метрика изражава путем једноставног бројања скокова. Ако су дозвољене веће вредности линка, тада је број скокова који могу бити употребљени неупоредиво мањи.

RIP је популаран протокол за рутирање због своје једноставности и добре прилагођености мањим подмрежама. Има својих ограничења и споменућемо само неке од њих:

- Како подмрежа расте, одредишта која захтевају метрику већу од 15 постају недоступна. Због тога RIP протокол не одговара већим мрежама;
- Једноставност метрике води до подоптималних табела рутирања. Резултат је да се пакети шаљу преко споријих (или на други начин захтевних) веза иако су доступне боље путање;
- Уређаји који не подржавају RIP протокол прихватиће RIP измене са било ког уређаја. Ово омогућава погрешно конфигурисаном уређају да лако поремети целу конфигурацију.

Рутирање помоћу протокола стања веза

У самом почетку се у ARPANET мрежи користио протокол за рутирање помоћу вектора растојања а од 1979. год. протокол за рутирање помоћу стања веза. Два најважнија недостака протокола за рутирање помоћу вектора растојања су:

- Алгоритам не узимао у обзир пропусни опсег линије приликом избора путања. У почетку су све линије биле на брзине 56 kb/s, па је нормално било водити само рачуна о броју линија које сачињавају путању до одредишта. Кад су почеле да се користе линије већих брзина од 230 kb/s и до 1.544 Mb/s не разматрати пропусни опсег је било неприхватљиво;
- Проблем бројање до бесконачности.

Из ових разлога, уведен је потпуно нови алгоритам, назван рутирање помоћу стања веза[1]. Разне варијанте овог алгоритма широко су и данас распрострањене. Основна замисао је једноставна и може се представити у пет целина. Сваки рутер мора да :

- открије своје суседе и сазна њихове мрежне адресе,

[1] *Link State Routing*

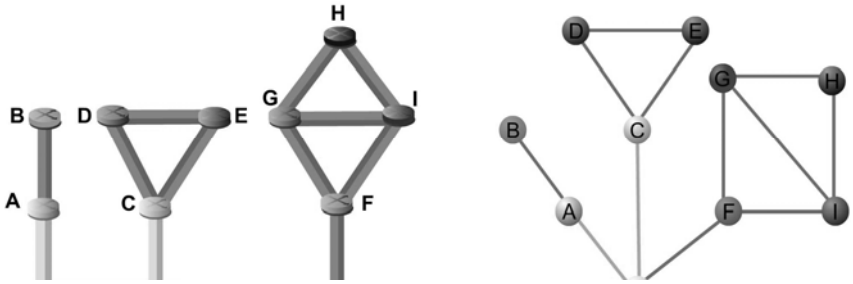
- измери кашњење, односно метрику до сваког суседа,
- формирати пакет који садржи информације које је сазнао,
- пошаље овај пакет свим другим рутерима,
- израчуна најкраћу путању ка сваком рутеру..

Откривање суседа

Када рутер почне с радом, његов први задатак је да сазна ко су му суседи. То рутер остварује слањем специјалног пакета - поздрава[1] на сваку тачка-тачка линију. Од рутера на другом крају се очекује да пошаље одговор са својим именом. Ова имена морају бити јединствена. Нпр. ако удаљени рутер добије информацију да су нека три рутера повезана са рутером F, веома је битно да може да се утврди да ли сва три мисле на исти рутер F.

Ако је више рутера спојено преко локалне рачунарске мреже ситуација је мало сложенија. Слика 5.6 приказује локалну рачунарску мрежу са којим су директно повезана три рутера: A, C и F. Сваки од ових рутера је повезан и са још неким рутерима (једним или више).

На слици 5.6 (б) локална рачунарска мрежа представљена је као нови, вештачки чвор N, с којим су спојени рутери A, C и F. Могућност преласка из рутара A у рутер C преко локалне мреже приказано је путањом ANC.



а) б)

Слика 5.6 (а) Девет рутера и локална рачунарска мрежа. (б) Граф модел слике (а).

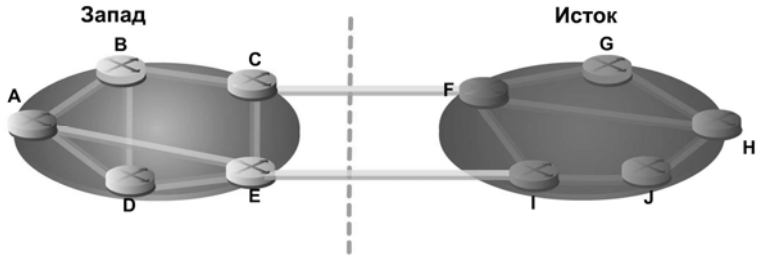
Мерење цене линије

Потребно је да сваки рутер зна кашњење или бар да има реалну процену кашњења до својих суседа. Непосредан начин да се одреди ово кашњење је слањем специјалног

[1] HELLO

пакета - оговора[1], који друга страна треба одмах да пошаље натраг. Када рутер време које је протекло од тренутка слања пакета -поздрава до тренутка приспећа пакета - одговора подели са два добије процену кашњења. Ако се жели тачнија процена, мерење времена се може бити извести више пута и користити се средња вредност. У овој методи подразумева се да су кашњења симетрична тј. иста у оба смера, што не мора да буде тачно.

Поставља се питање да ли треба приликом мерења кашњења урачунати и оптерећење мреже. Ако се урачунава и оптерећење мреже, часовник којим се мери време се мора покренути када пакет - одговор стане у ред. Ако се не узима у обзир оптерећење мреже, часовник се покреће када пакет-одговор доспе на чело реда.



Слика 5.7. Подмрежа у којој су источни и западни део повезани са две линије.

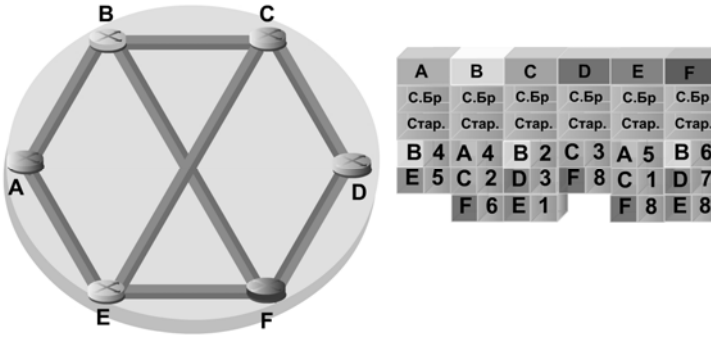
Ако се узима у обзир и оптерећење мреже, рутер ће при избору између две линије са истим пропусним опсегом одабрати ону која је мање оптерећена. На тај начин се побољшавају перформансе мреже. Понекад овај метод није тако ефикасан. Обратимо пажњу на подмрежу на слици 5.7, која је подељена у два дела, исток и запад, који су спојени двема линијама CF и EI. Претпоставка је да се већина саобраћаја одвија преко линије CF и као последица тога ова линија је веома оптерећена па су кашњења велика. Ако се при израчунавању оптерећење мреже узме у обзир најкраћа путања, линија EI ће постати привлачнија. Пошто су табеле рутирања ажуриране већина саобраћаја се сада одвија преко линије EI и она ускоро постаје преоптерећена. Као последица тога ће се у следећем ажурирању путања CF се појавити као најкраћа. Табеле рутирања ће осциловати, изазивајући непредвидиво и непоуздано рутирање. Ако се оптерећење мреже занемари и разматра само пропусни опсег линија до оваквих проблема не може да дође. Да би се избегле осцилације у избору најбољег путање добро би било да се саобраћај распореди на више линија, са одређеним делом предвиђеним за сваку појединачну линију.

[1] ECHO

Формирање пакета стања веза

Кад скупи све информације потребне за размену следећи корак сваког рутера је да направи пакет о стању везе. Пакет почиње идентификацијом пошиљаоца, следи секвенцијски број затим старост пакета и листа суседа. За сваког суседа уписано је кашњење до њега. Пример подмреже дат је на слици 5.8 (а). Ознаке на линијама представљају кашњења. Одговарајући пакети стања веза за свих шест рутера приказани су на слици 5.8 (б).

Прављење пакета стања веза је једноставно. Већи проблем је донети одлуку кад пакет треба направитиа. Једна од могућности је да се пакети периодично праве, тј. у стандардним временским интервалима. Друга могућност би била да се пакети праве када се догоди неки значајан догађај као што су: завршетак или почетак рада суседа, прекид или поновно успостављање линије итд.



Слика 5.8. Изглед подмреже и пакети стања веза за ову подмрежу.

Дистрибуирање пакета стања веза

Најделикатнији део алгоритма је дистрибуирање пакета стања веза на поуздан начин. Како се пакети дистрибуирају рутери који добију прве пакете и мењају своје путање. Може да се деси да рутери користити различите топологија мреже што доводи до недоследности, петљи, недоступних рачунара, итд.

Основна идеја лежи у коришћењу механизма преплављивања[1] за дистрибуцију пакета стања веза. Да би се алгоритам држао под контролом, сваки пакет садржи секвенцијски број који се инкрементира са сваким послатим пакетом. Рутери дуж путање

[1] *Flooding*

воде евиденцију о послатим пакетима за одређени изворишни рутер. Кад доспе нови пакет, у листи се проверава да ли је већ послат. Ако је није онда се пакет прослеђује на све линије сем на ону са које је стигао. Ако је дупликат онда се он одбацује. Ако пакет има секвенцијски број мањи од највишег дотад послатог одбацује као застарели.

Овај алгоритам има неколико решивих проблема:

ако секвенцијски бројеви крену од самог почетка због може да дође до прекорачења. Решење је користити 32-битни секвенцијски број. Ако се шаље један пакет стања веза у секунди биће потребно 137 година да се дође до прекорачење. На тај начин је могућност прекорачења искључена.

Други проблем је да ако рутер престане са радом изгубиће се траг о секвенцијским бројевима. Када рутер настави са радом и поново крене са секвенцијским бројем 0 пакет ће бити одбачени као дупликат.

Трећи проблем настаје ако дође до грешке у секвенцијском броју: на примиер секвенцијски број 65 540 уместо 4 (грешка у једном биту), пакети од 5 до 65 540 ће бити одбачени као застарели, јер је тренутни секвенцијски број грешком испао 65 540.

Овај проблем се може разрешити увођењем још једног поља које се означава као старост пакета и које се декрементира сваке секунде. Кад поље старост пакета достигне нулту вредност, пакет се одбацује. Вредност поља старост пакета се и при иницијалном процесу дистрибуције пакета декрементира у сваком рутеру. Ово је неопходно да би се обезбедило да се пакети не изгубе и евентуално опстану неодређени период времена.

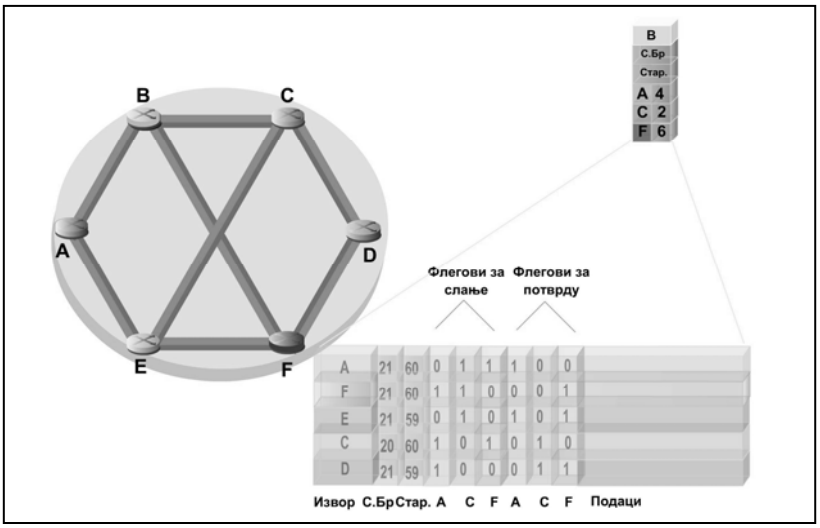
Неке преправке чине овај алгоритам робуснијим. Кад пакет протокола стања везе LSP[1] доспе у рутер за дистрибуцију он се не ставља одмах у ред за слање. Уместо тога, задржава се један кратак период времена у меморијском простору (баферу). Ако стигне други LSP пакет из истог извора пре него се пошаље прводоспели пакет, њихови секвенцијски бројеви се међусобно упоређују. Ако су једнаки дупликат се одбацује. Ако су различити, одбацује се старији пакет. Због заштите од грешака на линијама које повезују рутере за све LSP пакете се тражи потврда. Када је линија није активна меморисјки простор се прегледа скевенцијалним кружним редоследом[2] ради одабира пакета или потврде за слање.

Структура података коју користи рутер B за подмрежу на слици 5.8 приказана је на слици 5.9.

Сваки ред одговара скоро пристиглом и не потпуно обрађеним LSP пакетима. У табели је дато извориште пакета, секвенцијски број, старост, ознаке (флег) за слања и потврду. Ознака за слање постављена на вредност 1 указује да је потребно послати

[1] *Link State Protocol*
[2] *Round Robin*

пакета на одговарајућу линију. Када је постављена ознака потврде на вредност 1 онда то значи да треба послати LSP пакет потврде из рутера A стиже директно, треба га послати ка C и F и послати потврду рутеру A, што је и означено заставицама. Слично, пакет из F треба проследити ка A и C и послати потврду ка F. Но, ситуација са трећим пакетом, који шаље рутер E, је другачија. Пакет је стигао двапут, једном преко EAB и једном преко EFB. Дакле, треба га проследити само рутеру C, али потврдити и рутеру A и F.



Слика 5.9. Бафер за пакете рутера B са слике 5.8 (а).

Ако дупликат стигне док је оригинал још увек у баферу, битови се морају мењати. Нпр. ако копија LSP пакета рутера C доспе из F пре него се проследи четврти ред у табели, битови ће бити промењени у 100011, како би се означило да рутеру F треба послати потврду уместо пакета.

Израчунавање нових путања

Када рутер прикупи све LSP пакете, у ситуацији је да конструише читав граф подмреже, јер је свака веза представљена. Заправо, представљена је два пута, за сваки смер по једном. Ове две вредности се могу користити посебно или се може наћи средња вредност.

Потом се локално покреће *Dijkstra* алгоритам да се пронађу најкраће путање до свих могућих одредишта. Резултати ових израчунавања се инсталирају у табелама рутирања и наставља се са уобичајеним радом.

За подмрежу са *n* рутера, од којих сваки има *k* суседа, меморија потребна за складиштење улазних података је једнака производу $k \cdot n$. За велике подмреже ово може бити велики проблем. А и време израчунавања је критично. Свеједно, у многим практичним ситуацијама, *link state routing* ради веома добро. Имамо два битна фактора код протокола стања веза: захтеви за обрадом и меморијом и захтеви за опсегом.

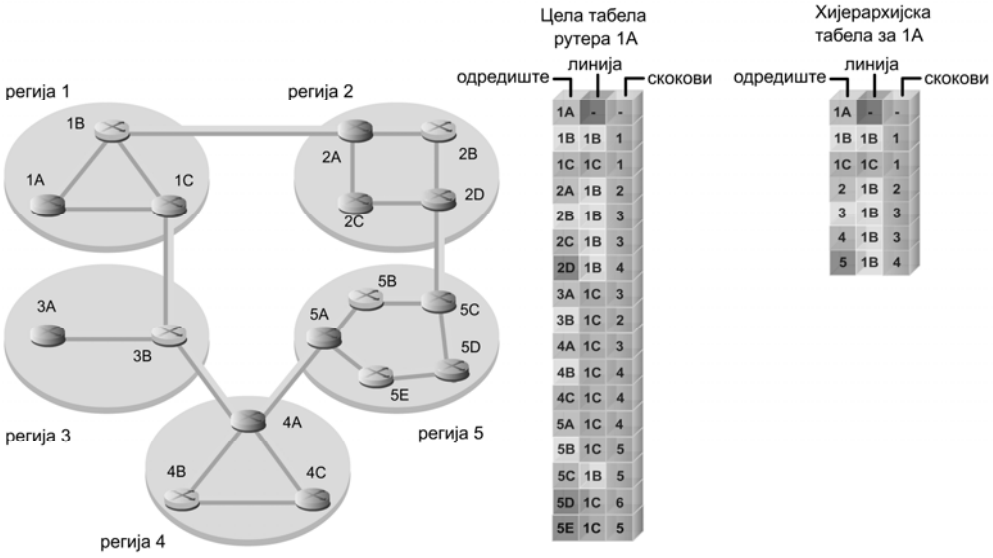
Коришћење протокола стања веза у већини случајева захтева више меморије и више времена се потроши на обраду података, него код протокола удаљеног вектора. Администратори мрежа морају обезбедити рутерима могућност резервације неопходних ресурса. Један део опсега се мора искористити за почетну размену пакета стања веза. У почетку сви рутери шаљу LSP пакете свим другим рутерима, што оптерећује мрежу и привремено смањује ширину опсега предвиђену за саобраћај корисничких података.

Како мреже расту, протоколи базирани на стању веза постају све атрактивније решење и то из следећих разлога:

- ови протоколи шаљу ажурирања само кад се топологија промени;
- периодична ажурирања дешавају се много ређе него код протокола са векторима удаљености;
- мреже у којима се користе протоколи базирани на стању веза могу да се поделе у хијерархијске области чиме се ограничава домет промена путања (више о овоме уследећем одељку);
- мреже у којима се користе ови протоколи подржавају бескласно адресирање;
- овакве мреже подржавају и резимирање (*summarization*).

Хијерархијско рутирање (*Hierarchical Routing)*

Како расту мреже, пропорционално расту и табеле рутирања. Не само да се троши меморија рутера на табеле које се непрестано повећавају, већ је потребно и више процесорског времена да их прегледа и више пропусног опсега за слање извештаја о њиховом стању. Када мреже нарасту до одређене тачке, више неће бити изводљиво да сваки рутер има запис о сваком другом рутеру, па рутирање треба спровести на хијерархијски начин, као у телефонској мрежи.



(а) (б) (в)

Слика 5.10. Хијерархијско рутирање.

У хијерархијском рутирању рутери су расподељени у регије и сваки рутер зна све детаље о рутирању пакета у оквиру своје регије, али не зна ништа о унутрашњој структури других регија. Када се различите мреже међусобно споје, природно је прогласити сваку посебном регијом, како би се рутери једне мреже ослободили потребе познавања топологије других мрежа.

Код великих мрежа двонивовска хијерархија може бити недовољна. Тад је потребно регије поделити у гроздове, гроздове у зоне, зоне у групе, итд. док постоји потреба да даљим дељењем. Као пример вишенивовске хијерархије, да видимо како се пакет из Крагујевца (Србија) рутира у Беркли (Калифорнија, САД). Рутер у Крагујевцу детаљно познаје топологију мреже у Србији, али саобраћај који је предвиђен да иде ван државе он шаље рутеру у Београду. Рутер у Београду је у могућности да проследи саобраћај другим центрима у регији, Атини, Тирани, Скопљу, Сарајеву, Загребу, итд, но он га прослеђује рутеру у Будимпешти, јер је он одговоран за прекоокеански саобраћај. Рутер у Будимпешти сав саобраћај за САД прослеђује у Њујорк. У Њујорку се саобраћај прослеђује у Лос Анђелес, који је задужен за прослеђивање саобраћаја унутар своје федералне јединице. И напослетку пакет стиже у Беркли.

На слици 4.10 (а) је приказано двонивовско хијерархијско рутирање са пет регија. Пуна табела рутирања рутера 1А има 17 записа. Ако се рутирање обавља хијерархијски, постоје записи за сваки локални рутер, а остале регије су представљене са по једним рутером, односно једним записом, тако да сав саобраћај за регију 2 иде преко линије 1В-2А, а остатак спољашњег саобраћаја иде преко линије 1С-3В. Хијерархијско рутирање смањило је табелу са 17 на 7 записа. Ове две табеле представљене су на сликама 4.10 (б) и 4.10 (в).

Но, ова уштеда у простору може довести до тога да се не одабира најповољнија рута. Нпр. видимо да је најбољи пут између регије 1 и 5 преко регије 2, но одређено је да сав саобраћај за регију 5 иде преко регије 3, јер је то повољније за већину одредишта у петици.

Код великих мрежа поставља се питање колико нивоа треба да имамо у хијерархији. *Kamoun* и *Kleinrock* (1979) су открили да је оптималан број нивоа за подмрежу са N рутера $\ln N$, а потребно је $e \cdot \ln N$ записа по рутеру. Такође су показали да је повећање ефективног средњег пута, проузрокованог хијерархијским рутирањем, довољно мало да је обично прихватљиво.

Рутирање емисијом[1]

[1] Broadcast Routing

У неким апликацијама, хостови треба да пошаљу поруке великом броју хостова или чак свима. Слање пакета свим одредиштима истовремено зове се слање један ка свима[1]. Постоје разне методе којима се то може учинити. Због једноставности, у даљем току рада користиће се термин емисија.

Један метод који не захтева специјалне карактеристике подмреже је једноставно слање различитог пакета сваком одредишту. Ово не само да троши много пропусног опсега, већ захтева и да извор има комплетну листу одредишта. У пракси може бити једина могућност, али је свакако најмање пожељна.

Преплављивање је још један метод који се може користити. Проблем са поплавом као техником емисије је исти као у случају тачка-тачка рутирајућег алгоритма: генерише превише пакета и троши много пропусног опсега.

Трећи алгоритам је рутирање ка више одредишта[2]. Сваки пакет садржи листу одредишта или битмапу са означеним одредиштима. Када пакет стигне у рутер, рутер проверава сва одредишта како би одредио скуп излазних линија које ће користити (излазна линија се користи ако је то најбоља путања до бар једног одредишта). Рутер генерише нову копију пакета за сваку излазну линију коју користи и укључује у сваки пакет само она одредишта за која користи ту линију. Дакле, скуп одредишта је подељен међу излазним линијама. Након довољног броја скокова, сваки пакет ће носити само једно одредиште и третираће се као обичан пакет. Вишеодредишно рутирање је слично одвојено адресираним пакетима, осим што кад неколико пакета следи исту путању, само један заузима пропусни опсег.

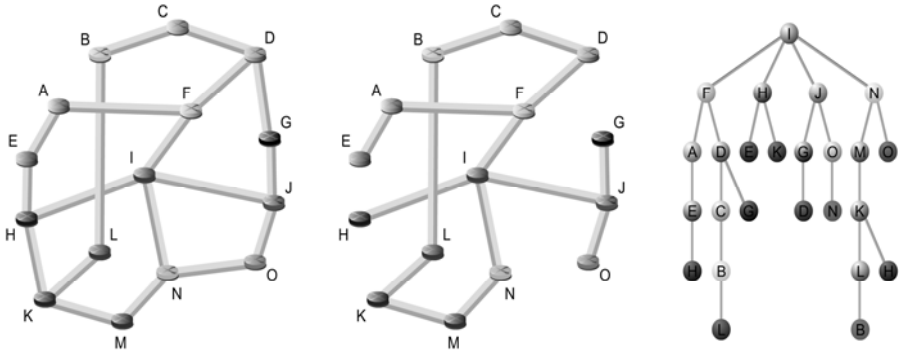
Четврти алгоритам користи *sink* стабло рутера који иницира емисију, или неко друго погодно *spanning* стабло. *Spanning* стабло је подскуп подмреже који укључује све рутере, али не садржи петље. Ако сваки рутер зна које његове линије припадају *spanning* стаблу, он може да копира пристигли емисиони пакет на све линије *spanning* стабла осим на ону са које је стигао пакет. Овај метод одлично користи пропусни опсег, генеришући минималан број пакета потребних за обављање посла. Једини проблем је што сваки рутер мора познавати структуру *spanning* стабла да би метод био применљив. Понекад је ова информација приступачна (нпр. *link state routing*), а некад не (нпр. *distance vector routing*).

Последњи алгоритам је покушај да се процени понашање претходног алгоритма, чак и ако рутери не знају баш ништа о *spanning* стаблу. Идеја, названа прослеђивање реверзном путањом (*reverse path forwarding*) је веома једноставна. Када емисиони пакет доспе у рутер, он проверава да ли је пакет стигао линијом која се обично користи за слање пакета извору емисије. Ако је тако, велике су шансе да је емисиони пакет такође

[1] дифузна емисија (*broadcasting*)

[2] (*multidestination routing*)

пратио најбољу путању и да је, према томе, прва копија пристигла у рутер. У овом случају рутер прослеђује његове копије на све линије сем на ону са које је стигао пакет. Ако је, међутим, емисиони пакет стигао другом линијом, пакет се одбацује као вероватни дупликат.



(а) (б) (в)

Слика 4.11. Прослеђивање реверзном путањом. (а) Подмрежа. (б) Sink стабло. (в) Приказ рада алгоритма.

Пример овог алгоритма погледајте на слици 4.11. Део (а) показује подмрежу, део (б) *sink* стабло, а део (в) приказује функционисање алгоритма. На првом скоку, рутер I шаље пакете ка F, H, J и N. Сваки од ових пакета стиже најбољом путањом до I и то је означено кругом око слова. На другом скоку, осам пакета се генерише, два од сваког рутера који је примио пакет у првом скоку. Како се испоставља, свих 8 стижу на досад непосећене рутере, а 5 од њих стиже најбољом путањом. Од 6 пакета генерисаних у трећем скоку, само 3 стижу најбољом путањом; остали су дупликати. После 5 скокова и 24 пакета, емисија се прекида. Четири скока и 14 пакета су пратили *sink* стабло.

Основна предност алгоритма *reverse path forwarding* је у томе да је истовремено веома ефикасан и лак за имплементацију. Не захтева се знање рутера о *spanning* стаблима. Нема претрпану листу одредишта, а ни битмапу у сваком одредишном пакету, као што то захтева вишеодредишно адресирање. Такође није потребан никакав специјални механизам за заустављање процеса, као код поплаве.

Рутирање вишеструким усмеравањем (*Multicast Routing)*

Неке апликације захтевају да прилично раздвојени процеси раде заједно у групи, нпр. група процеса која имплементира систем дистрибуираних база података. У оваквим ситуацијама, често је неопходно да један процес пошаље поруку осталим члановима групе. Ако је група мала, сваком члану може се послати тачка-тачка порука. Међутим, ако је група велика, ова стратегија је скупа. Понекад се може користити емисија, али користити емисију за обавештење 1000 рачунара на мрежи са милион чворова је неефикасно, јер већина чворова није заинтересована за поруку (или можда још горе; веома су заинтересовани, али не треба да је виде). Према томе, потребан је начин за слање порука прецизно дефинисаним групама, које имају мноштво чланова, али су мале у поређењу са мрежом као целином.

Слање поруке таквој групи зове се вишеструко усмеравање (*multicasting*). При том се користи алгоритам за рутирање вишеструким усмеравањем (*multicast routing*). За ову технику неопходно је управљање групама. Потребан је неки начин стварања и брисања група, као и начин да се омогући процесима да се прикључују групама, и да их напуштају. Кад се неки процес прикључи одређеној групи, веома је битно да обавести свог хоста о томе. Од велике је важности да рутери знају који од њихових хостова припада којој групи. Или хостови обавештавају рутере о променама својих група или рутери периодично испитују своје хостове. Рутери обавештавају своје суседе о томе, па се информације шире мрежом.

Да би се обавило овакво рутирање, сваки рутер израчунава своје *spanning* стабло. Нпр. на слици 4.12 (а) имамо две групе, 1 и 2. Неки рутери су прикључени на хостове који припадају једној или обе групе. *Spanning* стабло за крајњи леви рутер можете видети на слици 4.12 (б).

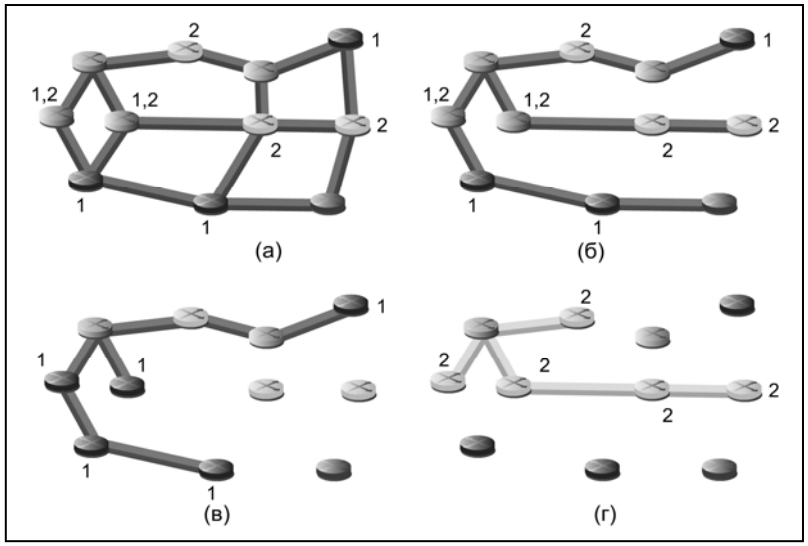
Кад процес пошаље групи пакет за *multicast*, први рутер прегледа своје *spanning* стабло и скраћује га, одстрањујући оне линије које не воде до хостова који су чланови групе. У нашем примеру, слика 4.12 (в) приказује скраћено *spanning* стабло за групу 1. Слично, слика 4.12 (г) приказује исто за групу 2. *Multicast* пакети се прослеђују само дуж одговарајућег *spanning* стабла.

Разни начини скраћења *spanning* стабла су могућа. Најједноставнији начин користи се код рутирања помоћу стања веза. Ту је сваки рутер свестан комплетне топологије мреже, укључујући и то који хостови припадају којим групама. *Spanning* стабло се скраћује, крећући од краја сваке путање према корену, уклањајући све рутере који не припадају траженој групи.

Код *distance vector routing* примењује се другачија стратегија. Основни алгоритам је *reverse path* forwarding. Кад год рутер без хостова заинтересованих за одређену групу и без веза са другим рутерима, прими *multicast* поруку за ту групу, он одговара са PRUNE поруком, којом даје до знања пошиљаоцу да му више не шаље поруке за ту групу. На овај начин, подмрежа се рекурзивно скраћује.

Један потенцијални недостатак овог алгоритма је да се слабо прилагођава великим мрежама. Претпоставимо да мрежа има *n* група, од којих свака има у просеку *m* чланова. За сваку групу, у меморији треба чувати *m* скраћених *spanning* стабала, што је укупно *mn* стабала. Ако постоји много великих група, потребан је огроман простор за чување свих информација.

Алтернативно, може се рачунати једно стабло по групи, са кореном (језгром) близу средишта групе (*core-based* стабла). Да би послао *multicast* поруку, хост је шаље језгру,



које то чини место њега. Иако није оптимално за све чворове, смањење меморијских трошкова са *m* стабала на једно стабло по групи је значајна уштеда.

Слика 4.12. (а) Мрежа. (б) Разгранато стабло[1] за крајњи леви рутер, (в) Стабло за групу 1[2].(г) стабло за групу 2.

Рутирање за мобилне кориснике[3]

Данас милиони људи имају преносне рачунаре. Они обично желе да читају свој е-пошту и приступе својим стандардним системима датотека, где год да су у свету. Ови мобилни хостови доносе нову компликацију у свет рачунарских мрежа; да би рутирала пакет мобилном кориснику, мрежа прво мора да га пронађе. Тема укључивања мобилних

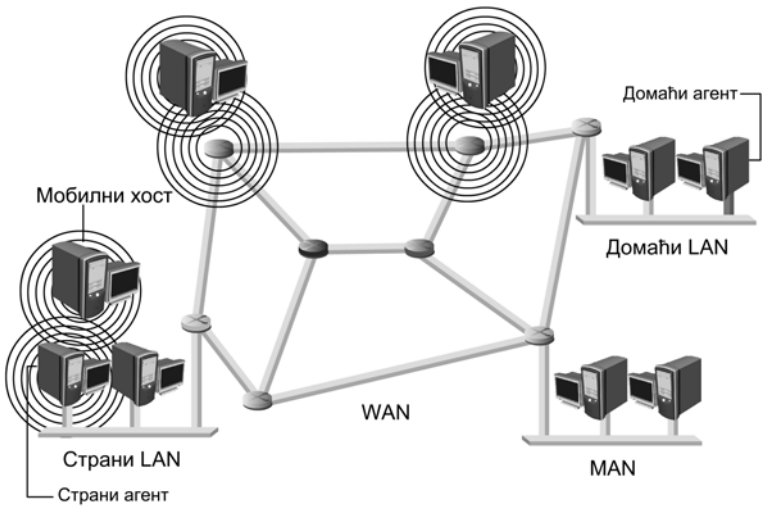
[1] Spanning
[2] Multicast
[3] *Routing for Mobile Hosts*

хостова у мрежу је веома свежа, но биће указано на неке проблеме и понуђена могућа решења.

Модел света који пројектанти мрежа често користе представљен је на слици 4.13. Ту је WAN, који чине рутери и хостови. LAN и MAN мреже су повезане с WAN мрежом, као и бежичне ћелије (*wireless cells*).

Хостове, који се никад не померају, назваћемо фиксним. Фиксни хостови су повезани са мрежом упреденим парицама или оптичким влакнима. Насупрот њима, можемо разликовати две друге врсте хостова. Миграцијски хостови (*migratory hosts*) су, у основи, фиксни хостови који се селе са једног места на друго с времена на време, али користе мрежу само кад се физички повежу с њом. *Roaming* хостови (*roaming hosts*) раде у покрету чувајући везу с мрежом током кретања. Користиће се израз мобилни корисници (*mobile hosts*) за две потоње категорије, тј. за све хостове који су одсутни од куће, а потребна им је веза с мрежом.

Претпоставља се да сви хостови имају сталну кућну локацију, која се никад не мења. Хостови такође имају сталну кућну адресу која се користи за одређивање кућне локације, слично као кад број телефона одаје којој држави и којој области у тој држави он припада. Циљ рутирања у системима са мобилним корисницима је омогућити слање пакета на кућне адресе и њихово допремање до корисника, ма где се они налазили.



Слика 4.13. WAN мрежа с повезаним LAN и MAN мрежама и бежичним ћелијама.

На моделу на слици 4.13, свет је подељен (географски) у мале јединице. Назовимо их областима, где је област обично LAN или бежична ћелија (*wireless cell*). Свака област има једног или више страних агената (*foreign agent*). То су процеси који воде евиденцију о свим мобилним хостовима који посећују област. Поред тога, свака област има и свог домаћег агента (*home agent*), који води евиденцију о хостовима који припадају тој области, а тренутно су у другој области.

Када нови хост уђе у област, рачунар се мора регистровати код страног агента те области. Процедура регистрације обично изгледа овако:

Периодично, сваки страни агент емитује пакет оглашавајући своје постојање и адресу. Придошли мобилни хост може чекати на једну од ових порука, али ако се ниједна не појави довољно брзо, може да емитује пакет питајући има ли неки страни агент у близини.

Мобилни хост се региструје код страног агента, дајући му своју кућну адресу, тренутну адресу слоја повезивања података и информације о сигурности.

Страни агент обавештава домаћег агента мобилног хоста о присуству његовог корисника у новој области, шаљући му поруку која садржи мрежну адресу страног агента, као и информације о сигурности, чија је улога да убеде домаћег агента да је мобилни хост заиста тамо.

Домаћи агент прегледа информације, које садрже и временски печат (доказ да су информације генерисане пре неколико секунди). Ако је све у реду, домаћи агент поручује страном агенту да настави с радом.

Када страни агент добије потврду од домаћег агента, он прави запис у својим табелама и обавештава мобилног хоста о успешној регистрацији.

Идеално, када хост напушта област требало би то да најави, како би се дозволила дерегистрација, но многи корисници нагло искључе своје рачунаре, кад заврше с радом.

Дакле, кад се пакет шаље хосту, рутира се ка његовој кућној адреси. Нпр. пошиљалац у Суботици жели да пошаље пакет преко пола Србије у Ниш. Пакете послате у Ниш пресреће домаћи агент у Нишу. Он претражује у својој бази нову локацију свог хоста и проналази адресу страног агента у Сегедину, којем је пријављен мобилни хост.

Домаћи агент потом чини две ствари. Прво, енкапсулира пакет у поље података другог пакета и шаље тај пакет страном агенту у Сегедин. Овај механизам се назива *tunneling*. Пошто прими пакет, страни агент узима оригинални пакет из поља података, енкапсулира га у рам *data link* слоја и шаље мобилном хосту.

Друго, домаћи агент обавештава пошиљаоца да убудуће пакете намењене овом хосту енкапсулира у поље података пакета, који се изричито адресирају на страног

агента, а уз то, наравно, шаље и адресу страног агента. Напослетку, пакети ће се рутирати до хоста директно преко страног агента.

Рутирање у *Ad Hoc* мрежама

Видели смо како изгледа рутирање у мрежама са мобилним хостовима и стабилним рутерима. Још драстичнија ситуација је она у којој су и рутери мобилни. Постоје следеће могућности:

Војна возила на ратишту без постојеће инфраструктуре.

Флота бродова на мору.

Радници хитне службе на месту земљотреса који је уништио инфраструктуру.

Скуп људи са преносним рачунарима у области која нема могућност повезивања бежичним мрежама (стандард 802.11).

У свим овим случајевима сваки чвор се састоји од рутера и хоста, обично на истом рачунару. Мреже чворова који се игром случаја налазе близу једни других називају се *ad hoc* мреже или MANET (*Mobile Ad hoc NETworks*).

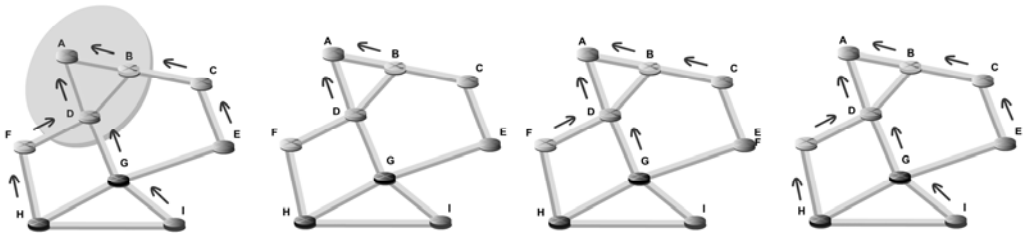
Оно што ове мреже разликује од фиксних мрежа је чињеница да сва правила о фиксним топологијама, стабилним и познатим суседима, јасном односу између IP адресе и локације, овде не важе. Рутери се могу појавити на новим местима и нестати сваког момента. Код фиксне мреже, ако рутер има важећу путању ка неком одредишту, та путања ће бити важећа неограничено дуго (сем ако дође до грешке негде у систему). Код *ad hoc* мрежа топологија се мења све време; пожељност и ваљаност путања може се мењати спонтано, без упозорења. Сувишно је и рећи да ове околности чине рутирање у *ad hoc* мрежама потпуно различитим од оног у фиксним мрежама.

На располагању је мноштво рутирајућих алгоритама за ове мреже. Један од интересантнијих је AODV (*Ad hoc On-demand Distance Vector*) алгоритам (*Perkins* и *Royer*, 1999). Овај алгоритам је сличан *Bellman-Ford* алгоритму са удаљеним векторима, али прилагођен да ради у покретном окружењу, да користи ограничен пропусни опсег и кратак батеријски век. Једна необична карактеристика је та да је у питању алгоритам на захтев, односно, он израчунава путању до неког одредишта тек онда кад неко жели да пошаље пакет ка том одредишту.

Откривање путање

У било ком тренутку, *ad hoc* мрежа може бити описана помоћу графа чворова (рутери + хостови). Два чвора су повезана (линијом у графу), ако могу директно комуницирати користећи своје радио везе. С обзиром да један чвор може имати јачи предајник од другог, могуће је да је A повезан са B, а да B није повезан са A. Но, због једноставности, претпоставићемо да су све везе симетричне. Треба нагласити и то да сама чињеница да

се два чвора налазе унутар радио опсега оног другог не значи да су повезани. Могу постојати зграде, брда и друге препреке које блокирају њихову комуникацију.



(а) (б) (в) (г)Слика 4.14. (а) Опсег емисије чвора A. (б) B и D су примили пакет чвора A. (в) C, F и G су примили пакет. (г) E, H и I су примили пакет. Стрелице показују могуће реверзне путање.

У циљу описа алгоритма, погледајмо мрежу на слици 4.14, где процес из чвора A жели да пошаље пакет чвору I. AODV алгоритам одржава табелу у сваком чвору (кључ табеле је одредиште), пружајући информације о одредиштима, укључујући и ону којем суседу треба послати пакете намењене неком од одредишта. Претпоставимо да A у својој табели не пронађе запис за I. Сад чвор A треба да открије путању до I.

Да лоцира I, A прави специјални ROUTE REQUEST пакет и емитује га (*broadcast*). Пакет доспева до B и D. Чворови B и D су повезани у графу са A, јер се налазе у његовом домету. F није повезан са A, јер не може примити радио сигнал који емитује A.

Формат ROUTE REQUEST пакета приказан је на слици 4.15. Садржи изворишну и одредишну адресу, типично IP адресе, које одређују ко кога тражи. Такође садржи идентификацију захтева (*Request* ID). То је локални бројач ког сваки чвор одржава посебно и инкрементира га сваки пут кад се ROUTE REQUEST емитује. Заједно, поље изворишне адресе и поље *Request* ID, јединствено идентификују ROUTE REQUEST пакет, омогућујући чворовима да одбаце дупликате које приме.

Слика 4.15. Формат ROUTE REQUEST пакета

Поред *Request* ID бројача, сваки чвор одржава и други секвенцијски бројач који се инкрементира сваки пут кад се пошаље ROUTE REQUEST (или одговор на нечији ROUTE REQUEST). Ради помало слично сату и користи се у сврху разликовања нових путања од старих. То четврто поље је секвенцијски бројач чвора A (односно изворишта у општем случају). Пето поље је најсвежија вредност секвенцијског бројача чвора I (односно

одредишта у општем случају), коју је A видео (0 у случају да је није видео). У даљем току текста ово ће бити јасније. Последње поље у пакету је бројач скокова, који води евиденцију колико је пакет направио скокова. Иницијализује се на вредност 0. Када ROUTE REQUEST пакет доспе у чвор (B и D у овом случају), обрада се одвија у следећим корацима:

Пар (изворишна адреса, ID захтева) се тражи у локалној табели историје ради провере да ли је захтев већ виђен и процесиран. Ако је дупликат, одбацује се и обрада се прекида. Ако није дупликат, пар се записује у табелу историје да би се убудуће дупликати могли одбацити, и обрада се наставља.

Пријемник тражи одредиште у својој табели рутирања. Ако је позната свежа путања (*fresh route*), шаље се ROUTE REPLY пакет до извора објашњавајући му како да стигне до одредишта. Свежа путања значи да је секвенцијски број одредишта смештен у табели рутирања већи или једнак секвенцијском броју одредишта у ROUTE REQUEST пакету. Ако је мањи, сачувана путања је старија од претходне путање коју је извор имао до одредишта. У том случају, извршава се корак 3.

С обзиром да пријемник не зна новију путању до одредишта, он инкрементира бројач скокова и реемитује ROUTE REQUEST пакет. Такође екстрактује податке из пакета и чува их као нови запис у својој табели реверзних путања (*reverse route table*). Ова информација се користи за конструисање реверзне путање, тако да се касније може послати одговор извору. Стрелице на слици 4.14 се користе за конструкцију реверзне путање. Покреће се бројач за нови запис реверзне путање. Ако истекне, запис се брише.

B и D не знају где је чвор I, па оба праве запис реверзне путање који упућује назад ка A, као што се види на слици 4.14, и емитују пакет са бројачем скокова постављеним на 1. Емисија из B досеже C и D. C такође прави запис у *reverse route table* и реемитује пакет. C друге стране D га одбацује као дупликат. Слично, *broadcast* чвора D одбија B. Но прихватају га F и G, и праве записе у својим табелама. Након што E, H и I приме пакет, ROUTE REQUEST коначно стиже до одредишта и сазнаје се путања. Иако су емисије овде приказане у три одвојена корака, емисије са различитих чворова нису координиране ни на који начин.

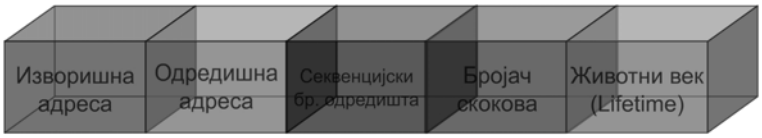
Као одговор на пристигли захтев, I прави ROUTE REPLY пакет (слика 4.16). Изворишна адреса, одредишна адреса и бројач скокова су копирани из ROUTE REQUEST пакета, али се бројач поставља на 0. Секвенцијски број одредишта се преузима из бројача у меморији. *Lifetime* поље контролише колико дуго је путања важећа. Овај пакет се шаље само чвору из којег је стигао ROUTE REQUEST пакет, у овом случају G. Тада следи реверзну путању ка D и коначно до A. У сваком чвору, бројач скокова се инкрементира тако да извор може да зна колико је одредиште удаљено.

У сваком чвору на путу назад, пакет се прегледа. Уписује се у локалну табелу рутирања путања до I, ако се испуне један или више од следећа три услова:

Није позната путања до I.

Секвенцијски број за I у ROUTE REPLY пакету је већи од вредности у табели рутирања.

Секвенцијски бројеви су једнаки, али је нова путања краћа.



Слика 4.16. Формат ROUTE REPLY пакета.

Овим поступком, као резултат открића путање чвора A, сви чворови на путањи ће знати путању до I. Чворови који су добили ROUTE REQUEST пакет, а не обухвата их реверзна путања (B, C, E, F и H), избацују запис о реверзној путањи по истеку бројача.

У великим мрежама, алгоритам генерише много емисија, чак и за одредишта која су у близини. Број емисија се може смањити на следећи начин. Бројач *Time to live* у IP пакету пошиљалац иницијализује на очекивани пречник мреже и он се декрементира после сваког скока. Кад досегне вредност 0, пакет се одбацује уместо да се емитује.

Могуће је и следеће решење. Да би лоцирао одредиште, пошиљалац шаље *broadcast* ROUTE REQUEST пакета са *Time to live* подешеним на 1. Ако се не добије одговор у неком очекиваном периоду времена, још један се шаље, овај пут са *Time to live* на 2. Следећи покушаји користе 3, 4, 5 итд. На овај начин, потрага се прво покушава локално, а потом се шири у све већим круговима.

Одржавање путање

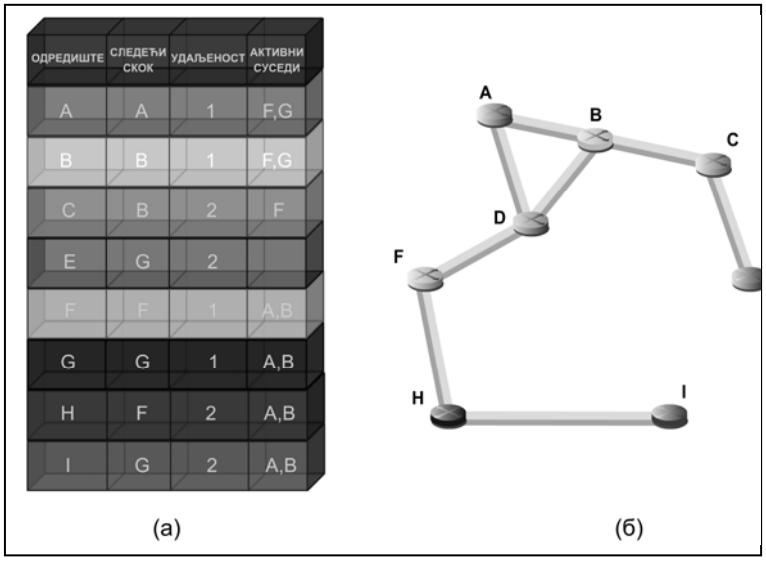
Чворови се могу кретати и искључивати, па се топологија може спонтано мењати. Нпр. ако се на слици 4.14 чвор G искључи, A неће знати да његова путања до I (ADGI) више није важећа. Алгоритам мора бити способан да се избори с тим. Периодично, сваки чвор емитује *Hello* поруку. Сваки сусед треба да одговори на њу. Ако нема одговора, пошиљалац зна да је сусед изашао из опсега радио таласа и да више нису повезани.

Ова техника се користи за одстрањивање путања које више нису у функцији. За свако могуће одредиште, сваки чвор (представићемо га са N) води евиденцију о својим суседима који су му послали пакет за то одредиште у последњих T секунди. Ови суседи су активни суседи чвора N за то одредиште. N има табелу рутирања (чији је кључ одредиште) која садржи излазни чвор који се користи да се доспе до одредишта, бројач скокова до одредишта, најсвежији секвенцијски број одредишта и листу активних суседа

за то одредиште. Могућа табела рутирања за чвор D у нашој топологији приказана је на слици 4.17 (а).

Када неки сусед чвора N постане недоступан, N проверава своју табелу рутирања да види које путање користе недоступног суседа. За сваку од ових путања, активни суседи се обавештавају да њихова путања преко N више није важећа и да је морају уклонити из својих табела рутирања. Активни суседи потом обавесте своје активне суседе, и тако рекурзивно, док се не уклоне све путање зависне од недоступног чвора.

Као пример одржавања путања, погледајте мрежу са слике 4.14, али сад без чвора G, који се изненада искључио. Промењена топологија се може видети на слици 4.17 (б). Кад D открије да нема чвора G, он претражује своју табелу рутирања и сазнаје да је G кориштен на путањама ка E, G и I. Унија активних суседа за ова одредишта је скуп {A, B}.Другим речима, A и B зависе од G на неким путањама, па их треба обавестити да ове путање више не раде. Чвор D им шаље пакете којима их обавештава да треба да ажурирају своје табеле на одговарајући начин. D такође избацује записе о путањама ка E, G и I из своје табеле рутирања.



Слика 4.17. (а) Табела рутирања чвора D пре отказивања чвора G. (б) Граф без G.

Можда није очигледно из описа, али критична разлика између AODV и *Bellman-Ford* алгоритма је у томе да чворови не шаљу периодичне емисије које садрже њихову комплетну табелу рутирања. Ова разлика штеди и пропусни опсег и батерије.

Трагање за чвором у мрежи равноправних рачунара (*Node Lookup in Peer-to-Peer Networks)*

Релативно новији феномен су мреже равноправних рачунара (тзв. *peer-to-peer networks*), у којима је велики број људи, обично са сталним фиксним везама са интернетом, у контакту због размене материјала. Прва распрострањена употреба *peer-to-peer* технологије је била за масовну осуду: 50 милиона *Napster* корисника је размењивало песме заштићене ауторским правом без дозволе власника ауторског права, све док *Napster* није судском одлуком затворен. У сваком случају, *peer-to-peer* технологија има и многе легалне и занимљиве употребе.

Оно што чини *peer-to-peer* системе занимљивим је чињеница да су потпуно дистрибуирани. Сви чворови су симетрични и нема централне контроле или хијерархије. У типичном *peer-to-peer* систему скоро сваки корисник има неке информације које би могле бити од користи другим корисницима. То може бити бесплатан софтвер, музика, фотографије, итд. Како доћи до неке информације? Добро решење јесте велика централна база података, али ово не мора бити изводљиво из неког разлога (нпр. нико није вољан да је хостује и одржава). Према томе, проблем се своди на то како да корисник пронађе чвор који садржи оно што он тражи у одсуству централизоване базе података или централизованог индекса.

Претпоставка је да сваки корисник има једну или више ставки, као што су фотографије, песме, програми, фајлови и друго, за које су други корисници заинтересовани. Свака ставка има *ASCII* стринг, који представља њено име. Потенцијални корисник зна *ASCII* стринг и жели да сазна да ли неко има ту ставку, и ако да, које су њихове IP адресе.

Као пример, размотриће се дистрибуирана генеалошка база података. Сваки генеалог има неке *on-line* записе за своје предаке и рођаке, вероватно са фотографијама, аудио и видео записима одређене особе. Више потомака има истог прадеду, тако да се записи о предаку вероватно налазе на више различитих чворова. Име записа је име особе у неком канонском облику. У једном тренутку, генеалог открива тестамент свог прадеде у архиви, у којем стоји да прадеда оставља свој златни сат нећаку. Генеалог сада зна нећаково име и жели да сазна да ли неки други генеалог има запис о њему. Како, без централизоване базе података, сазнати да ли неко има запис о нећаку, и ако има, ко?

Бројни алгоритми су предлагани за решење овог проблема. Овде је проучен *Chord* алгоритам. Поједностављено објашњење начина како он ради изложено је у даљем току текста. *Chord* систем састоји се од *n* учесника, а сваки од њих може имати неке записе, које је дужан да индексира ради других корисника. Сваки кориснички чвор има IP адресу, која се може хаширати у *m*-битни број помоћу хаш функције. *Chord* алгоритам користи *SHA*-1 за хаширање. То је функција која узима стринг варијабилне дужине и производи

160-битни број. Тако, можемо да конвертујемо било коју IP адресу у 160-битни број, који се зове идентификатор чвора.

Концепт је следећи: свих 2^{160} идентификатора чворова је поређано у растућем редоследу у великом кругу. Неки од њих одговарају стварним чворовима, а већина не. На слици 4.18 приказан је круг за *m*=5. У овом примеру, чворови са идентификаторима 1, 4, 7, 12, 15, 20 и 27 одговарају стварним чворовима и потамњени су на слици; остали не постоје.

Функција *successor(k)* даје као резултат идентификатор првог правог чвора који следи *k* у кругу у правцу казаљке на сату. Нпр. *successor*(6)=7, *successor*(8)=12, *successor*(16)=20.

Називи ставки (имена песама, имена предака, итд.) такође се хаширају истом хаш функцијом. Резултат се зове кључ (*key*). Дакле, да конвертујемо име (ASCII име ставке) у њен кључ, користимо *key=hash(name*). Ако особа жели да омогући другима да виде неку ставку, прво ће направити запис који садржи име ставке и сопствену IP адресу. Потом се позива *successor(hash(name))* да смести запис. Ако постоји више ставки (на различитим чворовима) са истим именом, њихови записи ће бити смештени у истом чвору.

Ако неки корисник жели да потражи неку ставку, хашираће име ставке да добије кључ и извешће функцију *successor(key)* да пронађе IP адресу чвора који чува запис о ставки. Да би било могуће пронаћи IP адресу чвора, сваки чвор мора одржавати одређене административне структуре података. Једна од њих је IP адреса следећег чвора дуж круга. Нпр. следбеник чвора 4 је 7, а чвора 7 чвор 12.

Значи, чвор са захтевом (онај који тражи одређену ставку) шаље пакет свом следбенику, који садржи IP адресу потражиоца и кључ који тражи. Пакет се шаље дуж круга док се не лоцира следбеник оног идентификатора чвора, чија вредност одговара вредности кључа. Чвор следбеник претражује да ли има неке информације које се подударају са кључем, и ако има, шаље их директно чвору потражиоцу, чију има IP адресу.

Као прва оптимизација, сваки чвор би могао чувати IP адресе и претходника и следбеника, тако да се претраживања могу слати и у једном и у другом смеру, зависно од тога за који се пут мисли да је краћи. Нпр. чвор 7 би могао ићи у потрагу за чвором 10 у правцу казаљке на сату, или у супротном смеру у потрагу за чвором 3.

Чак и са избором два смера потраге, линеарно претраживање свих чворова је веома неефикасно у великим *peer-to-peer* системима, јер је средњи број чворова (које је потребно обрадити) по потрази *n/2*. Да би се значајно убрзала претрага, сваки чвор одржава тзв. *finger table* (табелу показивача). Табела показивача има *m* записа, индексираних од 0 до *m*-1, а сваки показује на неки стваран чвор. Сваки запис има два

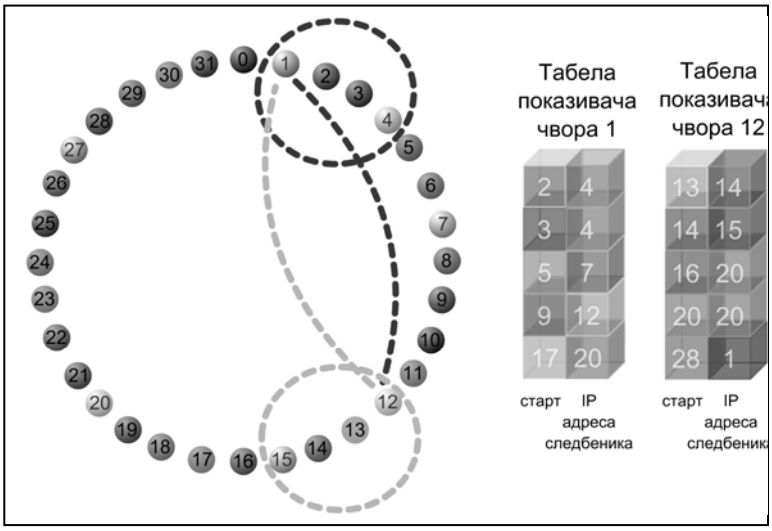
поља: старт и IP адреса следбеника старта *(successor(start))*. Вредности за ова поља у запису *i* чвора *k* су:

start = $k + 2^i$

IP адреса чвора *successor*(*start*[*i*])

Користећи табелу показивача, потрага за кључем у чвору *k* изгледа овако. Ако вредност кључа пада између *k* и *successor(k)*, онда следбеник чвора *k* држи запис о ставки и потрага је завршена. Ако то није случај, у табели показивача се тражи запис чије је поље старта најближи претходник вредности кључа. Захтев се потом шаље на IP адресу у том запису, где се наставља претраживање. Могло би се рећи да свака следећа претрага преполовљује преосталу раздаљину до циља. Може се показати да је просечан број претрага $\log_2 n$.

Ево једног примера потраге за кључем. Тражи се кључ 14, а потрага почиње у чвору 1. С обзиром да 14 не лежи између 1 и 4, консултује се табела показивача. Најближи претходник 14-ице у табели показивача је 9, па се захтев шаље на IP адресу у запису чвора 9, односно IP адресу чвора 12. Чвор 12 увиђа да се 14 налази између њега и његовог следбеника, 15, па враћа IP адресу чвора 1



(а) (б)

Слика 4.18. (а) Скуп 32 идентификатора у кругу. Светлији кругови одговарају стварним машинама (б) Примери табела показивача.

Чворови се све време прикључују мрежи и напуштају је. Систем мора пронаћи начин да ове операције држи под контролом. Када нови чвор, *r,* жели да се прикључи, он мора контактирати неки чвор у мрежи и питати га да израчуна *successor(r)* и врати IP адресу добијеног чвора. Потом, нови чвор пита свог будућег следбеника за IP адресу његовог претходника. Најзад, нови чвор објави овим чворовима да се убацује између њих у круг. Нпр. ако чвор 24 жели да се прикључи кругу, прво пита неки чвор да му провери *successor*(24), што је 27. Потом пита чвор 27 за његовог претходника, а то је чвор 20. После тога, објави обома свој улазак у круг. Чвор 20 користи 24 као свог следбеника, а 27 користи 24 као свог претходника. Уз то, чвор 27 предаје кључеве ставки у опсегу 21-24, који сад припадају чвору 24. У овој тачки, чвор 24 је потпуно прикључен мрежи.

Но, многе табеле показивача сад нису исправне. Због тога, сваки чвор извршава позадински процес, који периодично прерачунава сваки показивач позивом функције *successor*. Када неки од ових упита погоди нови чвор, одговарајући показивач се ажурира.

Кад чвор напусти мрежу регуларно, он предаје кључеве свом следбенику и обавештава свог претходника о свом одласку, тако да претходник може да се повеже са следбеником одлазећег чвора. Ако чвор ненајављено престане с радом, јавља се проблем, јер његов претходник више нема важећег следбеника. Како би се заобишао овај проблем, сваки чвор води евиденцију не само о свом директном следбенику, већ о својих *s* директних следбеника, дозвољавајући тако да се прескочи до *s*-1 узастопних покварених чворова и поново успостави круг.

Chord систем се користи за конструисање дистрибуираних датотечких система и других апликација, а истраживање је још у повоју.

закључак

Значај рутирања информација је изузетан у друштвима која функционишу и опстају на основу поуздане испоруке електронских информација. Рутирање представља главни процес којим се изводи пренос информација са једне на другу локацију. Чак и глобална економија је зависна од могућности рутирања информација из једног система у други.

Данашње мреже имају велики број рутера. Рутери преузимају податке у форми електронске поште, захтева за претраживање интернета, преноса датотека и њиховог испоручивања на одговарајућа одредишта. Електронска пошта се шаље на тачан сервер, захтева за претраживање интернета се прослеђује на интернет, а пернете датотеке се испоручују жељеном примаоцу. Све ове функције се заснивају на исправно постављеним и одржаваним рутерима.

6. УПРАВЉАЊЕ У РАЧУНАРСКИМ МРЕЖАМА

Како број рачунарских мрежа у оквиру једне организације расте заједно са разнородним системима који омогућавају њихово међусобно повезивање и повезивање на Интернет (рутери различитих произвођача, терминал сервер..) управљање овим системима постаје веома значајно. Постаје евидентно да:

- сама мрежа, њени ресуси и дистрибуиране апликације бивају од непроцењиве вредности за организацију,
- више ствари може да закаже као што је искључивање мреже или неког њеног дела као и деградирање перформанси до неприхватљивог нивоа.

Велике мреже не могу бити груписане и администриране само од стране људи. Комплексност таквих система диктира коришћење апликација за аутоматизовано управљање рачунарским мрежама. Потражња за таквим апликацијама се повећава, али такође постаје и све теже испоручити такве апликације, поготову ако се мрежа састоји од опреме различитих произвођача. Све већа децентрализација мрежних сервиса уз све већи значај радних станица и клијент/сервер система чини кохерентну и кординирану администрацију рачунарских мрежа све тежом. У тако комплексним информационим системима, многе битне ставке у мрежи су ван контроле њеног администратора.

Систем за управљање мрежом

Систем за управљање мрежом је колекција алата за надгледање и управљање мрежом:

- један интерфејс са моћним, али лаким за коришћење скупом команди за извршавање већине управљачких задатака,
- минимална количина додатне опреме тј., већина хардвера и софтвера потребног за управљање мержом укључено је у постојећу корисничку опрему.

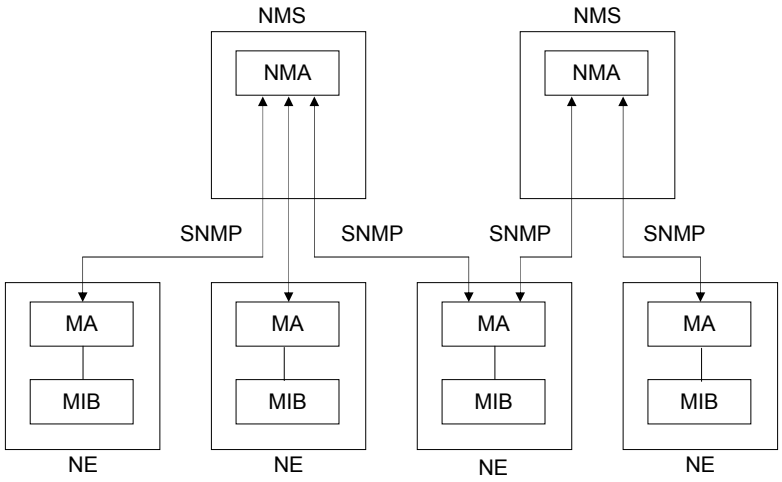
Систем за управљање мрежом састоји се од додатног хардвера и софтвера имплементираних у постојаће мрежне компоненте. Софтвер који се користи за извршавање задатака управљања мрежом налази се у крајњим станицама и комуникационим опреми (нпр. комутаторима, рутерима..). Систем за управљања мрежом пројектован је тако да види мрежу као јединствену архитектуру са адресама и ознакама

које су додељених свакој тачки и специфичним карактеристикама сваког елемента и сваке везе за коју систем зна. Активни елементи у мрежи редовно обезбеђују статусне информације управљачком центру мреже.

Протокол за управљање мрежом верзија SNMPv1

Протокол за управљање мрежом SNMP[1] развијен је као алат за управљање мрежама и међусобно повезаним мрежама које користе TCP/IP протокол. Касније је његова примена проширена и на сва остала мрежна окружења. Назив протокол за управљање мрежом указује на скуп спецификација за управљање мрежом која укључује и сам протокол као и дефиницију и концепт базе података. Модел управљања мрежом (слика 6.1) обухвата следеће кључне елементе:

- управљачка станица[2] (менаџер - станица),
- апликација за управљање[3],
- елеменат рачунарске мреже[4]
- агент[5],
- база података за управљање[6],
- протокол за управљање мрежом[7].



Слика 6.1 Компоненте система за управљање мрежом

[1] SNMP - *Simple Network Management Protocol*
[2] NMS - *Network Management Station*
[3] NMA - *Network Management Application*
[4] NE - *Network Element*
[5] MA - *Management Agent*
[6] MIB - *Management Information Base*
[7] *Network Managemnt Protocol*

Комуникација може бити двострана:

- менаџер тражи од агента одговор о специфичној променљивој. На пример колико је ICMP порука типа „порт је недоступан" је послао;
- агент обавештава менаџер да се нешто важно десило. На пример „мрежни интерфејс" је у квару;
- менаџер може да очита или постави вредност неке променљиве код агента На пример „промени подразумевану вредност поља TTL у IP заглављу на 64".

Управљачка станица је обично посебан, самосталан уређај али се може реализавати и као део другог система. У оба случаја управљачка станица служи као интерфејс човека-менаџера и управљачког система мрежа. Управљачка станица треба да садржи:

- скуп управљачких апликација које се користе за анализу података, опоравак од грешака итд.,
- интерфејс преко кога менаџер надзире и управља мрежом,
- могућност спровођења захтева менаџера мреже у процесу надзора и управљања удаљених елемената мрежне,
- базу података са информацијама о управљању мрежом добијених из база свих управљивих целина у мрежи.

Само последња два елемента су предмет SNMP стадардизације.

Други активни елемент у систему управљања мрежом је управљачки агент[1]. Кључни уређаји као што су станица, мост, комутатор и рутер могу бити опремљени са софтвером агената тако да њима може управљати станица-менаџер. Агент шаље управљачкој станици захтеване информације, извршава њене захтеве за управљањем а може и асинхроно снабдевати управљачку станицу информацијама од значаја.

Comment [A1]: Ја бих све енг. са фуснотама

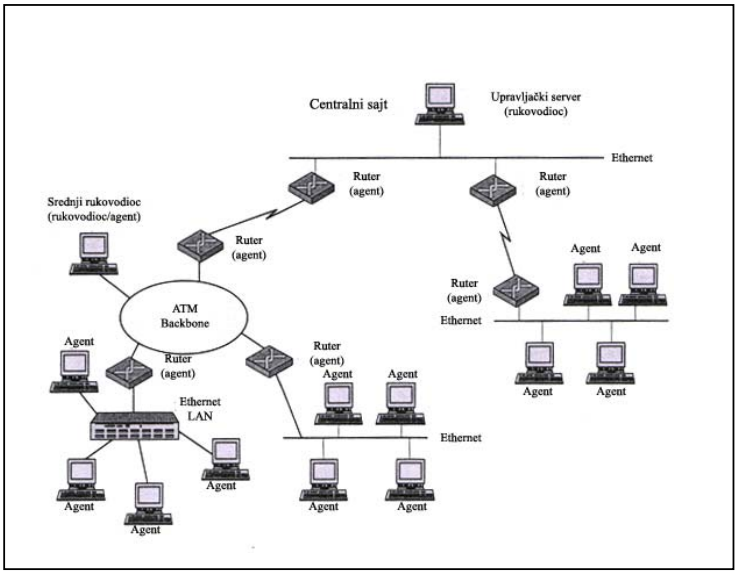
Да би контролисао ресурсе у мрежи, сваки ресурс је представљен као објекат. За објекат се може рећи да је то промељива која представља један аспекат управљачког агента. Колекција објеката назива се управљачка база података MIB. MIB функционише као колекција приступних тачака агента ка управљачкој станици. Ти објекти су стандардизовани за системе одређене класе (мостови подржавају исте управљачке објекте). Управљачка станица извршава функцију надгледања мреже тако што узима податке из MIB објеката. Управљачка станица може да управља агентима или може да промени конфигурационе параметре код агента тако што ће променити вредности одређених променљивих.

[1] *Management agent*

Управљачка станица и агенти су међусобно везани протоколом за управљање мрежом. Протокол који се користи за управљање у рачунарским мрежама са TCP/IP протоколом је једноставан протокол за управљање мрежом SNMP. Побољшана верзија овог протокола је означена са SNMPv2 и намењена је мрежама са TCP/IP и OSI архитектурама протокола. Сваки од ових протокола може да користи следеће операције:

- узми (*Get*) - омогућава управљачкој станици да добије вредности објеката са агената,
- постави (*Set*) - омогућава управљачкој станици да подеси вредности објеката код агената,
- обавести (*Notify*)- омогућава агенту да пошаље информацију управљачком систему о значајним догађајима.

У традиционално централизованој шеми управљања мрежом, једна крајња станица у конфигурацији има улогу мрежне управљачке станице; може да постоји још једна, или две управљачке станице које имају улогу резервних станица[1]. Други уређај садржи софтвер агента и MIB како би омогућио надзор и управљање станици - менаџеру. Како мрежа расте по величини и протоку података, такав централизован систем све више постаје немогућ за рад. Превише захтева се поставља пред управљачку станицу и превелик је проток података. Извештаји од сваког агента требају да пронађу свој пут кроз мрежу до главне станице. У таквим условима, децентрализовани, дистрибуирани приступ се показао као најбоље решење (слика 6.2).



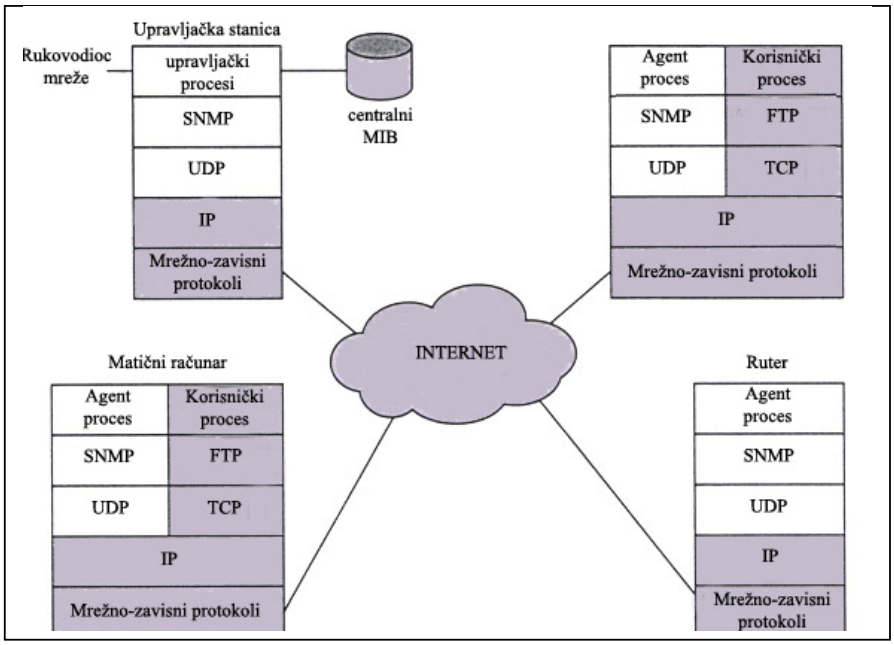
[1] *Backup*

Слика 6.2. Пример дистрибуиране конфигурације за управљање мрежом

У децентрализованој шеми управљања мрежом налази се више управљачких станица највишег нивоа[1] који се могу назвати и управљачки сервери[2]. Сваки такав сервер може директно да управља са одређеним бројем агената. Ипак, за многе агенте управљачки сервер даје задужења менаџеру средњег нивоа[3]. Менаџер средњег нивоа надгледа и контролише само оне агенте који су у његовој надлежности. Он такође има улогу и агента који обезбеђује информације и прихвата захтеве за управљањем од сервера-менаџера вишег нивоа. Оваква архитектуре распоређује оптерећење и смањује укупан проток у мрежи.

Архитектура протокола за управљање мрежом[1]

Протокол SNMP је на апликационом слоју и припада скупу TCP/IP протокола. Предвиђен је за рад са UDP протоколом. Слика 6.3 приказује типичну конфигурацију SNMPv1 протокола.



[1] *Top Level*
[2] *Management servers*
[3] *Intermediate manager*

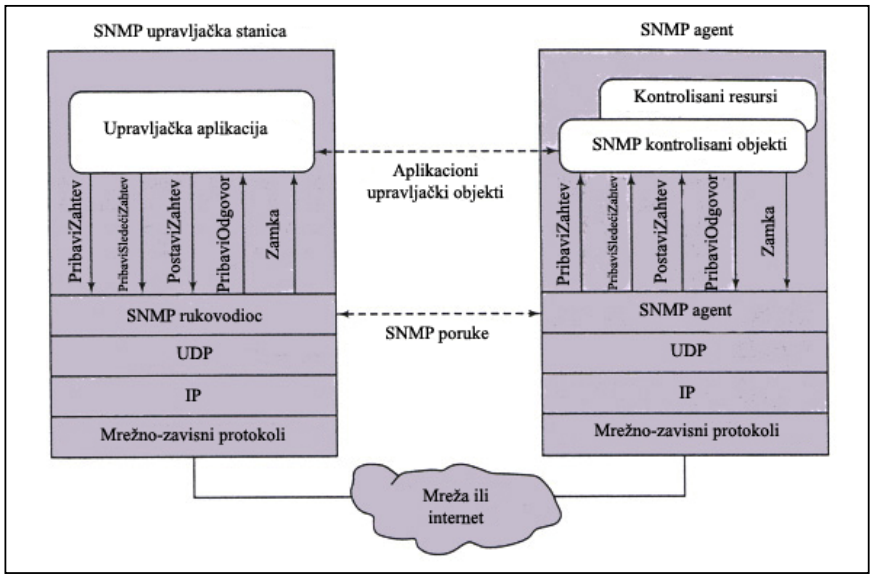
Слика 6.3. SNMPv1 конфигурација
За самосталну управљачку станицу, управљачки процес контролише приступ централној бази података MIB на страни управљачке станице и обезбеђује интерфејс према менаџеру мреже. Управљачки процес реализује управљање мрежом користећи SNMP протокол, који је постављен на самом врху изнад UDP, IP и осталих мрежних протоколе као што су Етернет, ATM[2] или штафетни пренос[3].

Сваки агент такође мора имати имплементиран SNMP, UDP и IP. Као додатак, постоји процес-агент који интерпретира SNMP поруке и контролише базу података MIB агента. Подршка протоколима као што су FTP, TCP као и UDP такође је потребан и код агената. На слици 6.3, осенчени делови приказују радно окружење са којим ће се управљати. Не осенчени делови омогућавају подршку функцијама управљања мрежом.

Слика 6.4 приказује детаљније виђење SNMP протокола. Апликација за управљање може да изда три врсте SNMP порука:

- узми захтев (*Get Request*),
- узми следећи захтев (*GetNextRequest*) и
- постави захтев (*SetRequest*).

[1] *Network Management Protocol Arhitetcture*
[2] *Asyncronous Transfer Mode*
[3] *Frame realay*



Слика 6.4. Улога SNMPv1 протокола

Прва два су две варијације функције узми (Get). Све три поруке су потврђене на страни агента у облику поруке узми одговор (*GetResponse*), која је прослеђена управљачкој апликацији. Као додатак, агент може да изда поруку (*Traps*[1]) као одговор на активност која утиче на MIB и остале контролисане ресурсе. Управљачки захтеви се шаљу на UDP порт 161 док агенти шаљу поруку-замку на UDP порт 162.

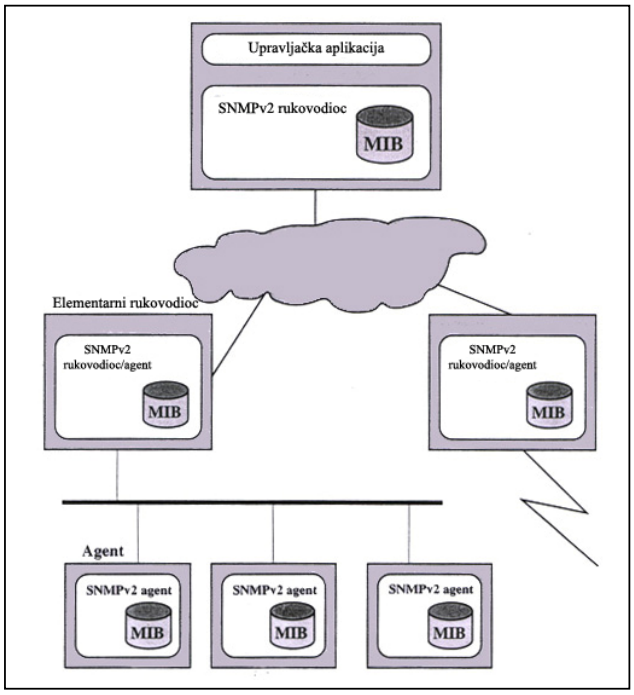
Протокол SNMP ослања се на UDP порт који је протокол без успоставе везе па је и сам SNMP протокол без успоставе везе. Пошто се не успостављања веза између управљачке станице и агената свака размена је посебна трансакција.

Протокол за управљање мрежом верзија SNMPv2

Августа 1988. год., издата је спецификације SNMP протокола и убрзо је постао доминантан стандард за управљање мрежом. Многи произвођачи понудили су самосталне радне станице за управљање мрежом базиране на SNMP протоколу и већина произвођача мостова, рутера, радних станица и персоналних рачунара понудила је SNMP агент пакете који омогућавају њиховим производима да их контролише SNMP управљачка станица. Као што и само име указује SNMP је једноставан алат за

[1] Замка

управљање мрежом. Он дефинише ограничену и лако применљиву базу података о управљању мрежом MIB састављену од скаларних променљивих и дво-димензионалних табела. Такође дефинише и аутоматизован протокол који омогућава менаџеру да прибави и подеси MIB променљиву и да омогући агенту да пошаље обавештење-замку. Једноставност је један од највећих адута протокола SNMP. Он се лако имплементира и не оптерећује много процесор а ни мрежне ресурсе. Такође структура протокола и MIB је довољно отворена, тако да није тешко постићи компатибилност између управљачких станица и агент софтвера различитих произвођача. Са његовим распрострањеним коришћењем недостаци SNMP су постали очигледнији. Недостаци се односе и на начин рада и механизме заштите. Последица је усавршена верзија позната као SNMPv2 коју су веома брзо произвођачи опреме прихватили и најавили уређаја са новом верзијом протокола за управљање мрежом. Верзија SNMPv2 не обезбеђује управљање мрежом већ оквир у коме се апликације за управљњем мрежом може направити. Апликације, као што су управљање грешкама, надгледање перформанси, наплата итд. су ван опсега овог стандарда. SNMPv2 обезбеђује инфраструктуру за управљање мрежом. Слика 6.5 пример је конфигурације која то илуструје.



Слика 6.5. SNMPv2 конфигурација за управљање

Суштина SNMPv2 је протокол који омогућава размену управљачких информација. Сваки учесник[1] у систему за управљање мрежом одржава локалну базу података са информацијама релевантним за управљање мрежом (MIB). SNMPv2 стандард дефинише структуру ових информација и дозвољене типове података и означава се као SMI[2]. Можемо да је опишемо као језик за дефинисање управљачких информација. Стандард такође обезбеђује већи број MIB врста[3] који су генерално веома корисни за управљање мрежом. Као допуна нове MIB врсте могу дефинисати произвођачи или групе корисника.

Барем један систем у конфигурацији мора бити одговоран за управљање мрежом. Може бити више управљачких станица како би се обезбедила редудантност или просто поделила задужења у великој мрежи. Већина осталих система има улогу агента. Агент прикупља информације локално и складишти их како би им касније приступио руководилац. Информације садрже податке о самом систему и могу садржати информације о мрежном протоку или мрежама на које је агент прикључен. SNMPv2 подржава и високо централизовану стратегију управљања мрежом и дистрибуирану стратегију. У последњим случајевима, неки системи имају двоструку улогу, управљачке станице и агента. У улози агента такав систем ће прихватити команде од вишег управљачког система. Неке од ових команди су везане за локални MIB код агента.

Остале команде захтевају од агента да се понаша ка заступник за удаљене уређаје. У таковом случају заступник агент преузима улогу менаџера како би приступио информацијама код удаљеног агента, и онда преузима улогу агента, како би проследио те информације вишем менаџеру.

Све ове размене се извршавају преко SNMPv2 протокола који је једноставан протокола типа захтев/одговор. SNMPv2 је обично имплементиран у на врху корисничког датаграм протокола UDP који је део TCP/IP референтног модела. Пошто се протоколом SNMPv2 размењује парови порука захтево/дговор, непрекидна и поуздана веза није потребна.

Структура управљачких информација

Структура управљачких информација SMI дефинише општи оквир унутар кога се дефинише и конструише база података MIB. SMI идентификује типове података који се могу користити у MIB бази података као и на који начин су ресурси у њој представљени и означени. Основна концепција структуре управљачких информација SMI је једноставност

[1] *Player*

[2] *Structure of management information*

[3] Термин MIB се може користити на више начина. Када се користи у једнини може да означи за базу података о информацијама за управљање на страни менаџера или на страни агента. Такође се може користи у једнини или множини када указује на специфичан скуп упрваљачких информација које су део свеобухватне базе података MIB. Тако SNMPv2 стандард у себи садржи дефиницију неколико врста MIB-а и укључује и MIB дефинисан са верзијом SNMPv1.

и могућност проширивања MIB базе података. Према томе, MIB може да складишти само просте типове података: скаларе и дводимензионе матрице (табеле). Структура управљачких информација SMI не подржава стварање и проналажење комплексних структура података. Овакав принцип није у складу са оним који се користи код система за управљање у оквиру OSI окружења код кога се сложена функционалност постиже са комплексним структурама података и сложеним претраживањем. SMI избегава комплексне типове података да би се поједноставила имплементације и рад са другим системима. MIB ће свакако садржати типове података које су креирали сами произвођачи и док се не поставе строга правила везане за дефинисање типова података међусобно повезивање система различитих произвођача (интероперабилност) неће бити могућа.

Три кључна елемента се налазе у SMI спецификацији. На најнижем нивоу SMI одређује типове података који могу да се складиште. Затим SMI одређује технику за дефинисање објеката и њихових табела. Коначно, SMI обезбеђује шему за повезивање јединствених ознака са сваким стварним објектом у систему тако да податке и агенте могу менаџери референцирати.

Табела 6.1 приказује типове података које SMI дозвољава. То је прилично ограничен скуп типова. На пример, реални бројеви нису подржани. Међутим, довољно је широк да подржи већину захтева у управљању рачунарском мрежом.

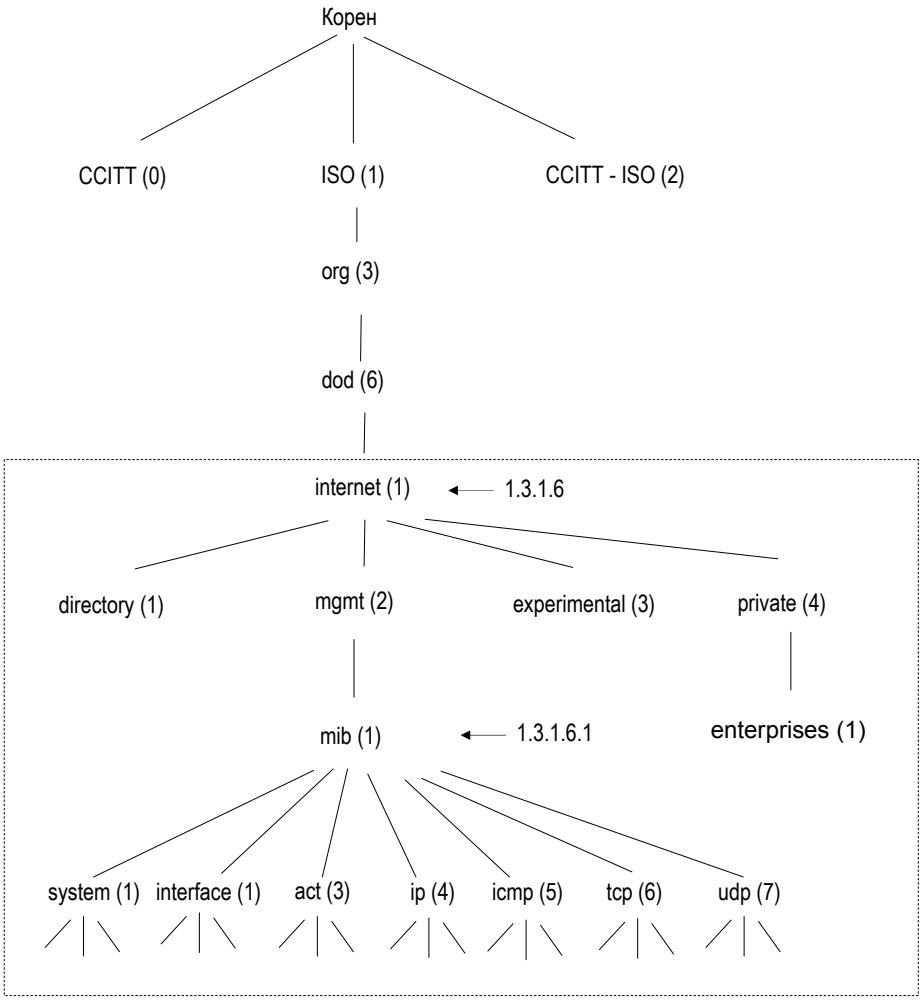
Тип податка	Опис
INTEGER	Цели бројеви у распону од 2^{31} до $2^{31} – 1$.
Uinteger32	Цели борјеви у распону од 0 до $2^{32} – 1$.
Counter32	Позитивни цели бројеви који могу бити увећани по модулу 2^{32}.
Counter64	Позитивни цели бројеви који могу бити увећани по модулу 2^{64}.
Gauge32	Позитивни цели бројеви који могу бити увећани или умањени, али не смеју прекорачити максималну вредност. Максимална вредност не може бити већа од $2^{32} – 1$. MIB варијабла tcpCurrEstab је пример: то је број TCP веза које су тренутно у стању ESTABLISHED или CLOSE_WAIT.
TimTicks	Позитивни цели бројеви који репрезентују време по модулу 2^{32}. На пример варијабла sysUpTime показује је време колоко дуго је агент подигнут изражено у стотинама секунди.
OCTET STRING	Низови октета за произвољне бинарне или текстуалне податке; могу бити ограничени на 255 октета.
IPAddress	OCTET STRING дужине 4, Интернет адресу.
PhysAddress	OCTET STRING одређује физичку адресу (нпр. 6 бајтова за Етернет адресу).
Opaque	Поље за произвољан бит.
BIT STRING	Списак именованих битова.

OBJECT IDENTIFIER	Идентификатор објекта. Административно додељено име (тип податка) објекту или другом стандардизованом елементу. Вредност је низ величине до 128 позитивних целих бројева.

Табела 6.1 Дозвољени типови података у SNMPv2 протоколу

Идентификатор објекта

Идентификатор објекта је низ целих бројева одвојених тачком. Бројеве спаја разграната структура (стабло) слична DNS стаблу. На самом врху стабла је неименовани корен од кога идентификатор објеата почиње. На слици 6.6 је структура стабла која се користи за системе са SNMP протоколом. Све променљиве у управљачкој бази MIB започињу са идентификатором објекта који почиње са 1.3.6.1.2.1



Интернет SMI

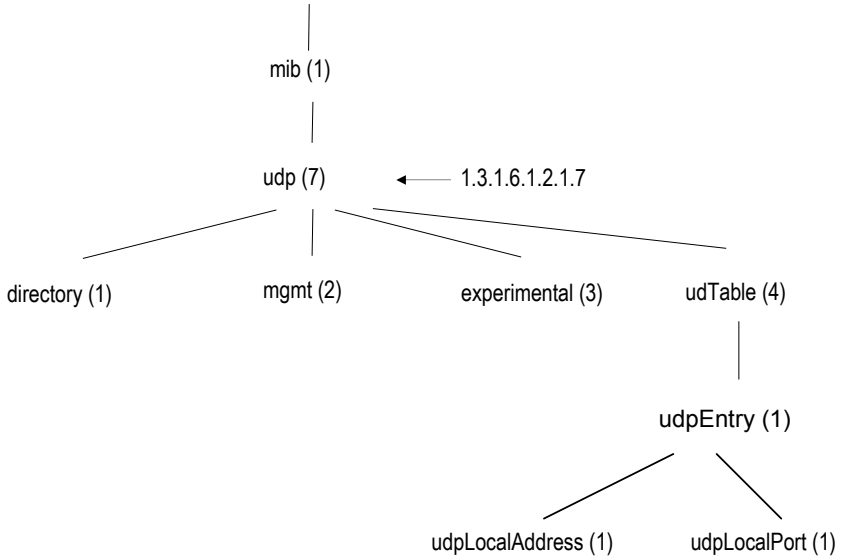
Слика 6.6 Идентификатори објеката у бази MIB

На слици 6.6 је поред MIB идентификатора објеката представљен и један именовани објекат: *iso.org.dod.inernet.private.enterprise* (1.3.6.1.4.1). То је место где се постављају MIB базе специфичне за одређеног произвођача. Овај чвор садржи листу од 400 регистрованих идентификатора који су додељени одговарајућим RFC документом.

База података MIB

База података за информацијама за управљање MIB је база података коју одржава агент а из које менаџер може да добије одговор (упит) или у коју менаџер може да постави одређене вредности. У овом поглављу биће дат кратак преглед базе података означене са MIB-II и описане у документу RFC1213.

Као што је на слици 6.6 представљено MIB је подељена у групе означене са: system, interfaces, at[1], ip итд. Описаћемо само променљиве из групе UDP. Ово је једноставна група са малим бројем promenqivih и једном табелом.



Слика 6.7 Структура стабла за табелу за IP адресу

Постоје четири променљиве и табела која садржи две променљиве. У табели 6.2 дат је опис променљивих.

Име	Тип податка	R/W	Опис
udpInDatagram	Counter		Број UDP датаграма који су испоручени корисничком процесу
udpNoPorts	Counter		Број примљених UDP датаграма за које није пронађен апликациони процес на одредишном

[1] *Address translation*

			порту
udpInErrors	Counter		Број неиспоручених UDP датаграма за које је неки други разлог а не "непостојање корисничког процеса" као што је нпр. грешка у контролној суми UDP датаграма.
udpOutDatagram	Counter		Број послатих UDP датаграма

Табела 6.2. Променљиве у групи *udp*

Колона означена са "R/W" је празна уколико се променљива може само очитати[1] или садржи оунаку (*) уколико се променљива може очитати и уписати[2]. Уколико је податак типа INTEGER са ограничењима у табели (нпр. 6.3) биће означена његова горња и доња граница. Две променљиве из *udpTable* су представљене у табели 6.3.

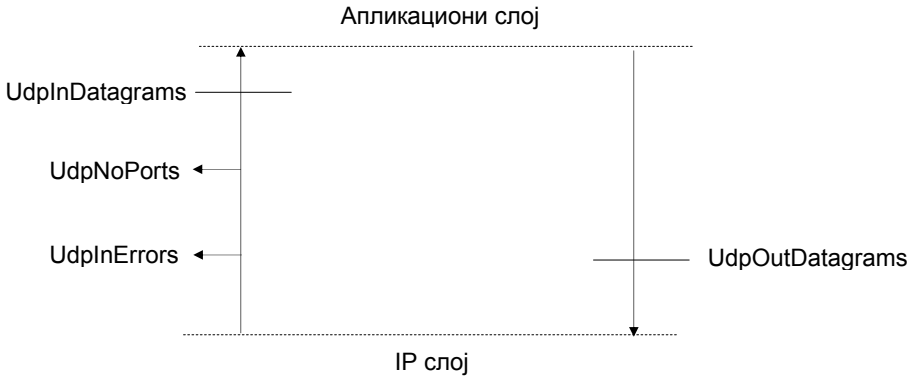
Табела UDP стране која ослушкује, индекс = <*udpLocalAddress*>. <*udpLocalPort*>			
Име	Тип податка	R/W	Опис
udpInDatagram	IpAddress		Локална адреса за страну која ослушкује. вредност 0.0.0.0 указује да је онај који ослушкује спреман да прихвати датаграм на било ком интерфејсу.
udpLocalPort	0...655351		Број локалног порта за оног који ослушкује

Тебела 6.3 Променљиве у *udTabl*e

Дијаграм стања

Постоји међусобна веза између прва три броајача[3] из табеле 6.2. Дијаграм стања[4] визуелно представља везу између различитих променљивих у MIB бази задате групе. Слика 6.8 представља дијаграм стања за UDP групу. Овај дијаграм показује да број UDP датаграма који се испоручују апликацији (*udpInDatagram*) једнак броју датаграма које IP слој испоручује транспортном слој (UDP) умањеним за број оних код којих је откривена грешка у контролној суми (*udpInErrors*) и оних за које није пронађена апликација на одредишном порту (*udpNoPorts*). Такође број UDP датаграма испоручених од IP слоја до транспортној слоја (*udpOutDatagram*) је број датаграма које апликација прослеђује транспорном слоју.

[1] *Read-only*
[2] *Read - Write*
[3] *Counter*
[4] [Case & Partridge 1989]



Слика 6.8. Дијаграм стања за UDP групу

Вредности идентификатора

Свака променљива у MIB бази мора бити индентификована када се SNMP рефернецира на њу, очитава или поставља њену вредност.Треба истаћи да се само чворови "листови" референцирају. SNMP не ради са одредишним колонама или редовима табела. Посматрајући слику 6.7 чворови "листови" су они који су описани на слици 6.8. Нису чворови "листови": *mib*, *udp*, *udpTable* и *udpEntry*.

Променљиве

Променљиве се рефернцирају додавањем "0" на идентификатор објекта променљиве. На пример бројач *udpInDatagram* из табеле 6.2 чији идентификатор објекта је 1.3.6.2.1.7.1 се референцира са 1.3.6.2.1.7.1.0. Текстуално име је:

iso.org.internet.mgmt.mib.udp.udpInDatagrams.0

Иако се име са којим се референцира променљива обично скраћује на *udpInDatagrams.0* треба нагласити да име променљиве које се јавља у SNMP поруци је идентификатор објекта 1.3.6.2.1.7.1.0.

Табеле

Вредност идентификатора записа табеле је много детаљнији. Погледајмо поново табелу стране која ослушкује (слика 6.7).

Један или више индекса је специфицирано у MIB бази за сваку од табела. За табелу стране која ослушкује MIB дефинише се индекс као комбинација две променљиве *udplocalaAddres* која је у IP address и *udpLocalPort* која је целобројана (индекс је постављен на почетку табеле 6.3). Претпоставимо да имамо три реда на UDP страни која ослушкује: први ред је за IP адресу 0.0.0.0 и порт 67, друга за 0.0.0.0 и порт 161 и трећа за 0.0.0.0 и порт 520 (табела 6.4).

Рад протокола

Главна компонента SNMPv2 оквира је сам протокол. Протокол обезбеђује директан, базичан механизам за размену управљачких информација између менаџера и агента. Основна јединица која се размењује је порука која је састављена од омотача и јединице протокола PDU[1]. Заглавље омотача је задужено за сигурност и касније ће бити више речи о њему. Јединица података протокола PDU може пренети седам врста SNMP порука. Општи формат је приказан на слици 6.8. Неколико поља је заједничко за више јединица података протокола PDU. Поље ознака-захтева[2] је цео број додељен тако да је сваки захтев јединствено означен. То омогућава менаџеру да међусобно повеже долазеће одговоре са захтевима који чекају на одговор.То такође омогућава агенту да разреши проблем дупликата јединица података протокола PDU настао као последица непоузданог мрежног сервиса (UDP). Поље повезивање-променљивих[3] садржи листу ознака објеката; зависно од јединице података протокола PDU, листа може такође да садржи вредност за сваки објекат.

Врста PDU[4]	Ознака-захтева	0	0	Повезивање-променљивих

а) Јединице података протокола PDU: *GetRequest*[5], *GetNextRequest*[6], *SetRequest*[7], SNMPv2-*Trap*[8], *InformRequest*[9]

Врста PDU	Ознака-захтева	Статус - грешке	Индекс - грешке	Повезивање-променљивих

б) *Response*[10]-PDU

[1] *Protocol data unit*
[2] *Request-id*
[3] *Variable-bindings*
[4] PDU type
[5] Узми захтев
[6] Узми следећи захтев
[7] Постави захтев
[8] Замка
[9] Захтев за информацијом
[10] Одговор

Врста PDU	Ознака-захтева	Нема понављања[1]	Максимум понављање[2]	Повезивање-променљивих

в) *GetBulkRequest*-PDU

Име1	Вредност1	Име2	Вредност2	***	Име*n*	Вредност*n*

г) Повезивање-променљивих

Слика 6.8. Форматјединице података протокола PDU за SNMPv2

Јединица података протокола *GetRequest* PDU коју издаје менаџер садржи листу једног или више имена објеката за које се тражи вредност. Ако је операција успешна одговарајући агент ће послати јединицу података протокола *Response.* Листа *Variable-bindings* ће садржати ознаке и вредности свих тражених објеката. За све променљива које нису релевантне за MIB, њихов ознака и порука о грешци враћају се у листи *Variable-bindings*. Дакле SNMPv2 не дозвољава делимичан одговор на *GetRequest***,** што је битно побољшање у односу на SNMPv1. У верзији SNMPv1 ако једна или више променљивих у јединици података протокола *GetRequest*-PDU није подржана агент ће вратити поруку о гешци која гласи: „нема таквог имена (*noSuchName*)". Да би се изборио са том грешком, руководиоц неће вратити никакву вредност или ће укључити такав алгоритам који ће када дође до грешке одстранити недостајућу променљиву променљиву поново шаљући захтев и прослеђујући апликацији непотпун резултат.

Јединицу података *GetNextRequest* PDU се такође издаје менаџер и садржи листу једног или више објеката. У овом случају за сваки објекат именован у *variable-bindings* пољу, вредност мора бити враћена за објекат који је следећи по лексикографском реду, тј. следећи по реду у структури стабла ознака објеката. Као што је било и са јединицом података протокола *GetRequest* PDU, агент ће вратити вредности за што је могуће више променљивих. Једна од способности јединицом података протокола *GetNextRequest* PDU је да омогући менаџеру да динамички претражује структуру MIB. Корисно је ако менаџер не зна унапред скуп објеката који подржава агент или који је у одређеној бази MIB.

Једно од главних унапређења у SNMPv2 је јединица података протокола *GetBulkRequest* PDU. Сврха овог јединице података протокола PDU је да смањи број размена у протоколу потребних да би се прибавила велика количина управљачких информација. *GetBulkRequest* PDU дозвољава SNMPv2 менаџеру да захтева да одговор буде што је могуће обимнији поштујући ограничење у величини поруке.

Јединицу података протокола *SetRequest* PDU издаје менаџер када захтева да се вредности једног или више објеката промени. SNMPv2 целина која прима поруку са јединицом података протокола *Response* PDU који садржи исту ознаку захтева. Код

[1] *Non-repeaters*
[2] *Max-repetitions*

SetRequest операција је или су све променљиве ажуриране или није ни једна. Ако је целина који одговар у могућности да постави вредности за све променљиве које се налазе у долазећој *variables-bindings* листи онда *Response* PDU поставља одговарајућа поља са вредностима које су за сваку од променљивих добијене. Ако се само једна од вредности променљиве не може добити не шаље се натраг ни један вредност и ни једна вредност се не ажурира. Онда статусни кôд о грешци указује на разлог за грешку и поље индекс-грешке указује на променљиву у листи која је довела до грешке.

Јединицу података SNMPv2-*Trap* PDU прави и шаље SNMPv2 целина који има улогу агента уколико дође до неубичајеног догађаја. Користи се да би управљачка станица са обавестила (асинхроно) о неким битним догађајима. Листа се користи да чува информације везане за *Trap* поруке. Јединице података протокола SNMPv2-*trap*-PDU за разлику од јединица података протокола *GetRequest* , *GetNextRequest*, *GetBulkRequest*, *SerRequest* и *InformRequest* PDU не захтева одговор од пријемног целине.

Јединицу података *InformRequest* PDU шаље SNMPv2 целина која је у улози менаџера једне апликације другој SNMPv2 целин која је у улози менџера да би прибавио управљачке информације за апликацију користећу ову другу целину. Као што је случај и са јединицом података SNMPv2-*trap* PDU, поље листе користи се да пренесе одговарајуће информације. Менаџер који прима *InformRequest* потврђује пријему са јединицом података протокола *Response* PDU.

За SNMPv2-*trap* и *InformRequest* важи да се различита стања могу одредити која указују када је обавештење направљено; такође је назначено и која се информација могу послати.

Протокол за управљање мрежом верзија SNMPv3

Многи недостаци у раду протокола за управљање SNMP су се задржали и у верзији SNMPv2. Да би се исправили недостатке SNMPv1 и SNMPv2 везани за сигурност направљена је нова верзија SNMPv3 која се појавила јануара 1998. год. и која је описана је документима RFC2570 до RFC2575. Скуп ових докумената не приказује комплетне могућности верзије SNMPv3 већ дефинише његову општу архитектуру и особености везане за сигурност. Намера је била да се све то искористи у оквиру верзије SNMPv2.

SNMPv3 обезбеђује три важна сервиса: аутентификацију, приватност и контролу приступа. Прва два су део корисничке сигурности USM[1] модела, а трећи је дефинисан у контроли приступа VABC[2] моделу. Сигурносним сервисима се управља преко

[1] *User-Based Security*
[2] *View-Based Access Control*

идентитета[1] корисника који захтевају сервисе. Овај идентитет се описује као принципал[2] а може бити особа или апликација, или група особа или апликација.

Механизам аутентификације код корисничке сигурности USM осигурава да је примљена порука послата од принципала чија се индентификација појављује у заглављу поруке. Овај механизам такође осигурава да порука није промењена у путу и да није намерно закашњена или поново направљена. Предајни принципал обезбеђује аутентификацију стављајући аутентификациони кôд поруке у SNMP поруку коју шаље. Овај кôд је функција садржаја поруке, идентитета предајног и пријемног пара, времена преноса и тајног кључа који треба да знају само пошиљаоц и примаоц. Тајни кључ мора бити подешен ван корисничке сигурности USM као конфигурацина функција. Менаџер конфигурације или мреже је задужен за дистрибуцију тајног кључа и убацивање у базе података различитих SNMP менаџера и агената. То може да се уради физички или користећи неку форму сигурног преноса податак изван USM. Када пријемни принципал добије поруку он користи исти тајни кључ да би израчунао исти аутентификациони кôд поруке. Ако верзија кôда код пријемника одговара вредности додатој долазној поруци, онда пријемник зна да порука потиче од ауторизованог принципала и да порука није промењена током пута. Дељени тајни клуч између предајног и пријемног пара мора бити раније подешен. Тренутни аутентификациони кôд који се користи је познат као HMAC, који је стандардан механизам аутентификациј на Интернету .

Део који се односи на за приватност у USMу омогућава менаџерима и агентима да шифрују поруке. Опет, менаџер принципал и агент принципал морају да деле тајни кључ. У овом случају, ако су две стране конфигурисане да користе приватност сав саобраћај између њих ће бити шифрован преко помоћу DES[3]алгоритма. Предајни принципал шифрира поруку користећи DES алгоритам и његов тајни кључ и шаље поруку пријемном принципалу који је дешифрује помоћу DES алгоритма и истог тајног кључа.

Деоа за контролу приступа пружа могућност конфигурисања агената тако да пруже различите нивое приступа бази управљачких информација MIB агента различитим менаџерима. Агент принципал може да ограничи приступ својој бази управљачких информација MIB одређеном менаџер принципалу на два начина:

- Први начин је да му ограничи приступ на само одређен део бази управљачких информација MIB. На пример, агент може да ограничи већини менаџера приступ само делу за преглед статистике перформанси, а само једном одређеном менаџеру омогући преглед и ажурирање конфигурационих параметара;

[1] *Identity*
[2] *Principal*
[3] *Data Encryption Standard*

- Други начин је да агент може да ограничи активности које менаџер може да изврши на одређном делу базе управљачких информација MIB. Рецима одређени менаџер принципал може само очитава[1] неки део базе управљачких информација MIB агента. Полиса контроле приступа која се користи од стране било ког агента, за сваког принципал мора бити претходно конфигурисана и састоји се од табеле у којој се налазе детаљи права приступа различитих ауторизованих менаџера.

[1] *Read-only*