Глава 14: Multicast Sockets*

(у вези са главом 13: UDP Datagrams and Sockets)

Сокети из поглавља 13 су unicast: обезбеђују point-to-point комуникацију.
Unicast сокети креирају конекцију између две добро дефинисане крајње тачке;
постоји један пошиљалац и један прималац и, иако они могу заменити улоге, у сваком тренутку једноставно је рећи ко је ко. Међутим, иако point-to-point комуникација задовољава многе, тј. већину потреба (људи користе 1-1 конверзације миленијумима), многи задаци захтевају другачији модел.

Нпр. **телевизијска станица** одашиље податке са једне локације у сваку тачку унутар опсега свог предајника. Сигнал долази до сваког ТВ-а, било да је он укључен или не и да ли је подешен баш на ту станицу. **Ово је типичан пример *broadcasting*-а**. Он је недискриминишући и прилично трађи електромагнетски спектар и енергију.

Насупрот томе, видеоконференција шаље аудио-видео **одабраној групи** људи.
Ажурирања DNS рутера путују од њега обзнањујући промену многим другим рутерима.
Међутим, **пошиљалац се узда да ће посредници копирати и одаслати поруку даље**.
Он **не адресира поруку на сваки хост** који ће је напослетку примити.
Ово су примери *multicasting*-а, иако имплементирани додатним апликационим слојем протокола поврх TCP или UDP. Ови протоколи захтевају прилично детаљно конфигурисање и људску интервенцију. Међутим, скорашњи развој мрежног софтвера у већини оперативних система као и Интернет рутери отварају нову могућност - прави multicasting, у коме **рутери одлучују како да ефикасно проследе поруку појединачним хостовима**. Посебно, **иницијални рутер шаље само једну копију поруке рутеру близу одредишних хостова, који затим прави већи број копија за различите примаоце у или близу одредишта**. Интернет multicasting је изграђен **поврх UDP-а**.
Multicasting у Јави користи класу `DatagramPacket` и класу `MulticastSocket`.

Шта је Multicast сокет?

Multicasting је више од unicast point-to-point комуникације, али мање и циљаније од broadcast комуникације. Multicasting шаље податке од једног хоста већем броју различитих хостова, али не свима, **подаци иду само клијентима који су изразили интересовање приступајући одређеној multicast групи**. На неки начин ово је *као јавни скуп*. Људи могу доћи и отићи по сопственом нахођењу, одлазећи када их дискусија више не занима. Пре него што пристигну и након што оду, уопште не морају да процесирају информације, оне до њих и не допиру. На Интернету, такви „јавни скупови“ се најбоље имплементирају коришћењем multicast сокета који шаље копију података локацији (или групи локација) у близини учесника који су изразили интересовање за те податке. У најбољем случају, подаци се копирају само када стигну у локалну мрежу која услужује заинтересоване клијенте: подаци пролазе Интернетом само једанпут. Реалистичније, неколико идентичних копија података обилази Интернет; али пажљивим одабиром тачака у којима се токови дуплирају оптерећење мреже је минимизовано. **Добра вест** је да програмери и мрежни администратори нису одговорни за одабир тих тачака, чак ни за слање већег броја копија; тиме рукују **Интернет рутери**.

*** поједине чињенице важиле су у тренутку писања књиге, више не важе**

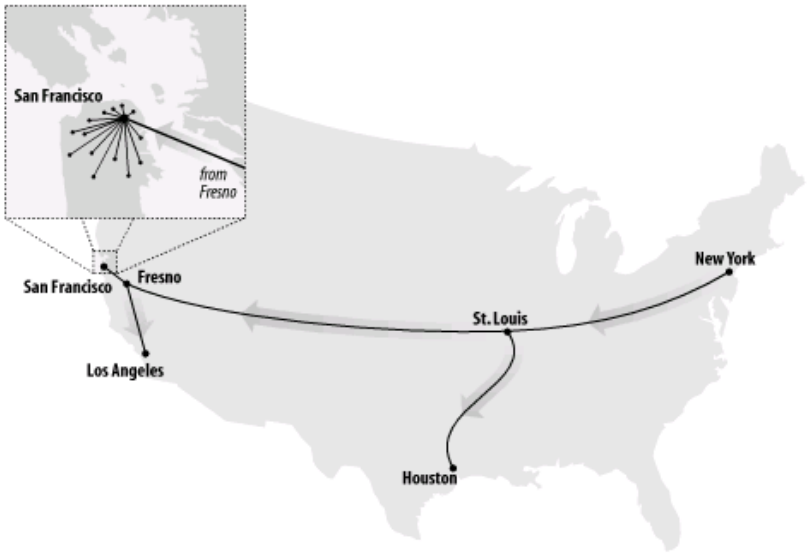
IP такође подржава broadcasting, али је његова употреба строго ограничена. Протоколи захтевају broadcast само када не постоји алтернатива, а рутери ограничавају broadcast на локалну мрежу или подмрежу, спречавајући да крене Интернетом. Чак и неколико малих глобалних broadcast-а може бацити Интернет на колена. Broadcasting података попут аудија или видеа или чак велике количине текста и слика не долази у обзир. Један једини email spam који иде на милионе адреса је довољно лош. Замислите шта би се догодило када би се видео копирао свим 600 милиона Интернет корисника (у тренутку писања књиге!) желели они да га гледају или не.

Међутим, постоји и нешто између point-to-point комуникације и broadcast-а целом свету. Нема разлога слати видео хостовима који нису за њега заинтересовани; потребна је технологија која шаље податке хостовима који их желе, не узнемиравајући остатак света. Један начин да се то постигне је користити много unicast токова. Ако 1000 клијената жели неке податке, они се шаљу хиљаду пута. То је неефикасно, пошто се подаци непотребно дуплирају, али је редовима величине ефикасније од broadcasting-а података сваком хосту на Интернету. Ипак, ако је број заинтересованих клијената довољно велик, на крају можете остати без протока или процесорске снаге - вероватно пре него касније.

Други приступ проблему је креирање статичких стабала конекције (енг. **connection trees**). Овакво решење користи **Usenet news** и неки конференцијски системи (**CUseeMe**). Подаци иду од изворишног места ка другим серверима, који их реплицирају за друге сервере, који их евентуално реплицирају за клијенте. Сваки клијент се конектује на најближи сервер. То је ефикасније од слања свега свим заинтересованим клијентима преко већег броја unicast-а, али ова шема почиње да застарева. Нове локације морају ручно да нађу место за убацивање у стабло. Дрво не мора нужно да одражава најбољу могућу топологију у сваком тренутку, а сервери ипак морају да одржавају много point-to-point конекција са својим клијентима, шаљући исте податке свакоме.
Било би боље дозволити рутерима на Интернету да динамички одреде најбоље могуће руте за слање дистрибуиране информације и да реплицирају податке само када је то апсолутно неопходно. Ту на сцену ступа multicasting.

Нпр. ако „мултикастингујемо“ видео из Њујорка и 20 људи прикачених на један LAN гледа шоу у Лос Анђелесу, feed ће бити послат том LAN-у само једанпут. Ако још 50 људи гледа у Сан Франциску, data stream ће бити негде дуплиран (нпр. у месту Fresno) и послат за Сан Франциско и Лос Анђелес. Ако још 100 људи гледа у Хјустону, још један data stream ће бити послат тамо (можда из St. Louis-a):

Figure 14-1. Multicast from New York to San Francisco, Los Angeles, and Houston



Подаци иду Интернетом само три пута - не 170 пута што би било потребно за point-to-point конекције или милионима пута што би био случај код правог broadcast-a. Multicasting је на пола пута између point-to-point комуникације уобичајене за Интернет и broadcast модела телевизије и ефикаснији је од оба. **Када је пакет multicast, адресиран је на multicast групу и послат сваком хосту који припада тој групи.** Он не иде појединачном хосту (као код unicasting-a), нити сваком хосту (као код broadcasting-a). Било шта од тога било би превише неефикасно.

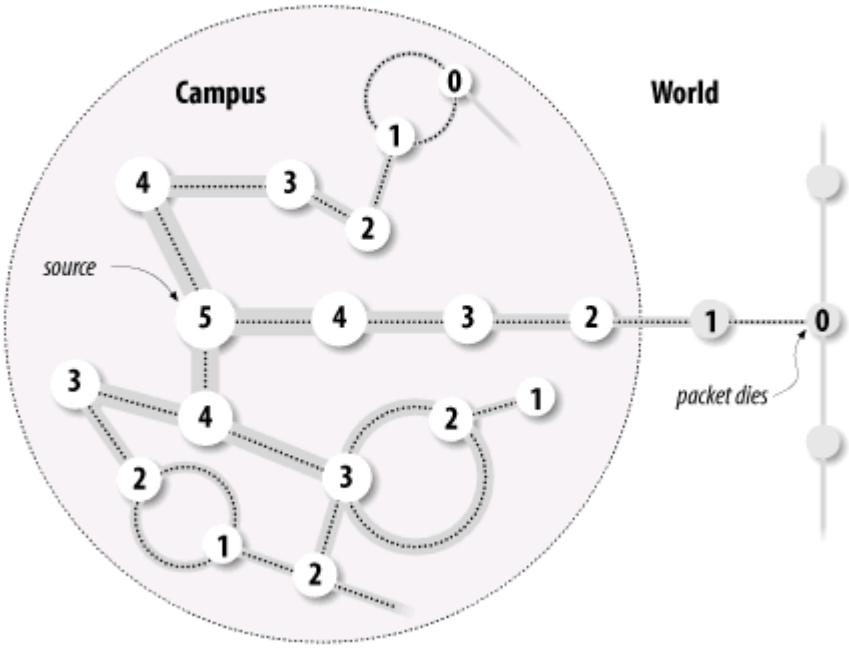
*Када људи почну разговор о multicasting-у, **аудио и видео** су прве апликације на које се помисли, међутим, то је само врх леденог брега. Друге могућности укључују **multiplayer игре**, **дистрибуиране фајл системе**, **massively parallel computing**, **multiperson conferencing**, **database replication**, итд.*
*Multicasting се може користити за **имплементирање пате сервиса и directory сервиса** који не захтевају да клијент унапред зна адресу сервера; како би дошао до имена, хост може урадити multicast свог захтева познатој адреси и чекати док не стигне одговор од најближег сервера.*
***Apple-ов Ranezvous** (a.k.a. Zeroconf) и **Sun-ов Jini** користе IP multicasting како би динамички открили сервисе локалне мреже.*

Multicasting такође олакшава имплементацију разних врста кеширања за Интернет.
Мартин Хамилтон је 1995. предложио коришћење multicasting-а за изградњу дистрибуираног серверског система за World Wide Web. Нпр. оптерећени web сервер се може поделити на већи број машина, од којих све деле један hostname, мапиран на multicast адресу. Претпоставимо да једна машина рукује HTML фајловима, друга сликама, а трећа процесира сервлете. Када клијент пошаље захтев на multicast адресу, захтев се шаље сваком од три сервера. Када сервер прими захтев, он гледа да ли клијент жели HTML фајл, слику или одговор сервлета. Ако сервер може да обради захтев, он одговара. Иначе, сервер игнорише захтев и препушта другим серверима да га процесирају. Лако је замислити комплексније поделе задужења између дистрибуираних сервера.

Multicasting је дизајниран да се уклопи у Интернет што је могуће боље. **Већину посла раде рутери. Апликација просто шаље datagram пакет на multicast адресу, која се суштински не разликује од било које друге IP адресе. Рутери обезбеђују да се пакет испоручује свим хостовима у multicast групи. Највећи проблем је што multicast рутер још увек није свеприсутан**; према томе неопходно

је знати довољно о њима како би се установило да ли их локална мрежа подржава. Што се тиче саме апликације, неопходно је обратити пажњу на **додатно поље заглавља у datagram-има, Time-To-Live (TTL) вредност**. TTL је максимални број рутера који datagram може да прође; када досегне тај број, одбацује се. Multicasting користи TTL као ad hoc начин да ограничи **колико далеко пакет може путовати**. Нпр. не желите да пакети за пријатељску игру у кампу доспеју до рутера на другом крају света. Следећа слика илуструје како TTL-ови ограничавају ширење пакета

Figure 14-2. Coverage of a packet with a TTL of five



Multicast адресе и групе

Multicast адреса је адреса коју дели група хостова која се назива **multicast група**.
За почетак причамо о адреси. Multicast адресе су IP адресе из опсега 224.0.0.0 до 239.255.255.255. Све адресе из овог опсега имају бинарне цифре 1110 као своја прва 4 бита. Називају се адресама *Класе D* како би се разликовале од уобичајенијих адреса класа A, B и C. Као и свака IP адреса, multicast адреса може имати hostname; нпр. multicast адреси 224.0.1.1 (адреса Network Time Protocol дистрибуираног сервиса) придружено је име ntp.mcast.net.

Multicast група је скуп Интернет хостова који деле multicast адресу. **Сви подаци послати multicast адреси отпремају се свим члановима групе. Чланство у multicast групи је отворено; хостови могу ући и напустити групу у произвољном тренутку. Групе могу бити сталне или привремене.** Сталним групама придружују се адресе које остају исте без обзира да ли има или нема чланова у групи. Међутим, већина multicast група је **привремена**, постоје само док имају чланове. **Све што је потребно урадити је креирати нову multicast групу са произвољном адресом од 225.0.0.0 до 238.255.255.255, конструисати `InetAddress` објекат као адресу и почети са слањем података тој групи.**

Известан број multicast адреса одвојен је за специјалне сврхе.
all-systems.mcast.net, 224.0.0.1, је multicast група која укључује све системе који подржавају multicasting на локалној подмрежи. Група experiment.mcast.net, 224.0.1.20, се обично користи за локално тестирање. (Не постоји multicast адреса која шаље податке свим хостовима на Интернету). Све адресе које почињу са 224.0.0 (тј. адресе од 224.0.0.0 до 224.0.0.255) резервисане су за протоколе рутирања и друге активности ниског нивоа, попут откривања gateway-а и извештавања о чланству групе. Multicast рутери никада не прослеђују datagram-е са одредиштима из овог опсега.

IANA је задужена за руковање сталним multicast адресама. За сада, придружено их је неколико стотина*. Већина их почиње са 224.0, 224.1, 224.2 или 239. Табела 14-1 у књизи приказује неколико ових перманентних адреса. Неколико блокова адреса величина од неколико туцета ☺ (десетина) до неколико хиљада адреса такође је резервисано за специјалне сврхе. Потпуна листа доступна је на http://www.iana.org/assignments/multicast-addresses. Преосталих 248 милиона адреса класе D свако коме треба може користити на привременој основи. Multicast рутери (скраћено mrouters) одговорни су да обезбеде да два различита система не покушавају да користе исту адресу класе D у исто време.

Клијенти и сервери

Када хост жели да пошаље податке multicast групи, он смешта податке у **multicast datagram-е, који нису ништа друго до UDP datagram-и адресирани на multicast групу**. Већина multicast података је аудио или видео или обоје. Ова врста података тежи да буде релативно велика и релативно робусна за губитак података. Ако се током преноса изгуби неколико пиксела или чак читав frame видеа, сигнал је и даље препознатљив. Према томе, **multicast подаци се шаљу преко UDP-а**, који, иако непоуздан, може бити више од 3 пута бржи него TCP. (Ако размислите, multicast преко TCP био би немогућ. TCP захтева да хостови потврде да су примили пакете; руковање потврдама пријема у multicast ситуацији било би ноћна мора.) Уколико развијате multicast апликацију која не може толерисати губитак података, Ваша је одговорност да утврдите да ли су подаци оштећени у путу и како руковати изгубљеним подацима. Нпр. ако градите дистрибуирани систем за кеширање, можете просто одлучити да фајлове који не стигну неоштећени оставите ван кеша.

TTL вредност је један бајт у заглављу који узима вредности од 0 до 255; грубо се интерпретира као број рутера кроз које пакет може проћи пре него што буде одбачен. Сваки пут када пакет прође кроз рутер, TTL поље се декрементира најмање за 1; поједини рутери могу га декрементирати за 2 или више. Када TTL достигне 0, пакет се одбацује. TTL поље је оригинално смишљено да спречи петље гарантујући да ће сви пакети евентуално бити одбачени; оно спречава лоше конфигурисане рутере да бесконачно једни другима шаљу пакете напред и назад. У IP multicasting-у, TTL ограничава multicast географски. Нпр. TTL вредност 16 ограничава пакет на локалну област, у општем случају на једну организацију или можда организацију и њене непосредненадређене и подређене суседе. TTL вредност 127, међутим, шаље пакете широм света. Могуће су и међувредности. Међутим, не постоји прецизан начин за мапирање TTL-ова у географска растојања. Уопштено, што је место удаљеније, то више рутера пакет мора да прође док не стигне на одредиште. Пакети са малим TTL вредностима неће путовати толико далеко као они са великим. Табела 14-2 приказује грубе процене. **Пакети адресирани на multicast групу од 224.0.0.0 до 224.0.0.255 никада се не прослеђују изван локалне подмреже, без обзира на коришћене вредности TTL-а.**

Table 14-2. Estimated TTL values for datagrams originating in the continental United States

Destinations	TTL value
The local host	0
The local subnet	1
The local campus—that is, the same side of the nearest Internet router—but on possibly different LANs	16
High-bandwidth sites in the same country, generally those fairly close to the backbone	32
All sites in the same country	48
All sites on the same continent	64
High-bandwidth sites worldwide	128
All sites worldwide	255

Након што су подаци смештени у један или већи број datagram-а, хост шаље datagram-е преко Интернета. То је као слање регуларних (unicast) UDP података. Одашиљући хост стартује слањем multicast datagram-а локалној мрежи. Пакет непосредно стиже до свих чланова multicast групе у истој подмрежи. Ако је поље TTL пакета веће од 1, multicast рутери локалне мреже прослеђују пакет другим мрежама које имају чланове одредишне групе. Када пакет стигне на неку од одредишних дестинација, multicast рутер „стране“ мреже одашиље пакет до сваког хоста који опслужује, а који је члан multicast групе. Ако је потребно, multicast рутер такође одашиље пакет наредним рутерима на путевима између текућег рутера и свих евентуалних одредишта.

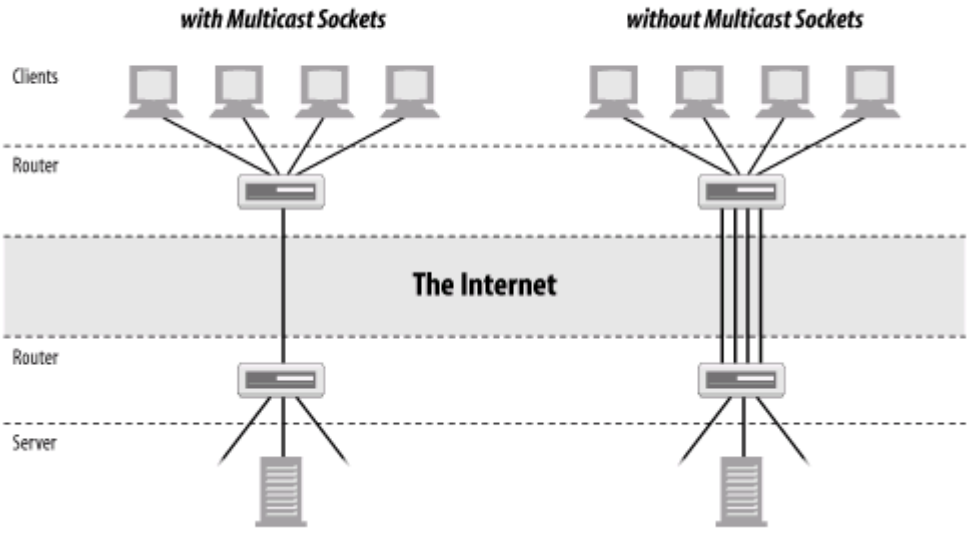
Када подаци стигну до хоста у multicast групи, хост их прима као што прима било који UDP datagram - иако одредишна адреса datagram-а не одговара хосту примаоцу. Хост препознаје да му је datagram намењен јер припада multicast групи на коју је datagram адресиран. Хост прималац мора ослушкивати на одговарајућем порту, спремном да процесира datagram када пристигне.

Рутери и рутирање

Следећа слика приказује једну од најједноставнијих могућих multicast конфигурација: један сервер шаље исте податке четворици клијената опслуживаних истим рутером. Multicast сокет шаље један ток података преко Интернета рутеру клијената; рутер умножава ток и шаље га сваком од клијената. Без multicast сокета, сервер би морао да шаље 4 одвојена, али идентична, тока података рутеру, који би рутирао сваки ток клијенту. Коришћење истог тока за слање истих података већем броју клијената значајно смањује проток кроз костур Интернета.

Наравно, руте у стварности могу бити много комплексније, укључујући вишеструку хијерархију редундантних рутера. Међутим, циљ multicast сокета је једноставан: без обзира колико је мрежа комплексна, исти подаци никада не би требало да буду послати више од једанпут преко било ког сегмента мреже. На срећу, ми не морамо да бринемо о проблемима рутирања. Само креирамо `MulticastSocket`, придружимо га multicast групи и сместимо адресу multicast групе у `DatagramPacket` који желимо да пошаљемо. Рутери и класа `MulticastSocket` се брину о свему осталом.

Figure 14-3. With and without multicast sockets



Највеће ограничење на multicasting је доступност специјалних multicast рутера.
То су реконфигурисани Интернет рутери или радне станице које подржавају IP multicast екстензије. Многи потрошачки оријентисани ISP-ови намерно не омогућују multicasting у својим рутерима. 2004, још увек је било могуће наћи хостове између којих не постоји multicast рута (тј. не постоји рута између хостова која путује искључиво преко multicast рутера).

Да би се multicast подаци слали и примали изван локалне подмреже, неопходан је multicast рутер. Проверите са својим мрежним администратором да ли ваши рутери подржавају multicasting. Можете такође пробати ping-овање all-routers.mcast.net. Ако било који рутер одговори, Ваша мрежа је повезана на multicast рутер:

>> ping all-routers.mcast.net

all-routers.mcast.net is alive

Ово још увек не мора да значи да је могуће слати или примати од сваког multicast-способног хоста на Интернету. Да би Ваши пакети стигли до сваког датог хоста, мора постојати пут multicast-способних рутера између Вашег и удаљеног хоста. Алтернативно, поједине локације могу бити конектоване специјалним multicast tunnel софтвером који одашиље multicast податке преко unicast UDP-а који разумеју сви рутери. Уколико имате потешкоће са примерима из поглавља како бисте добили очекиване резултате проверите са својим локалним мрежним администратором или ISP да видите да ли је multicasting заправо подржан Вашим рутерима.

Рад са Multicast сокетима

У Јави, подаци се multicast-ују користећи класу `java.net.MulticastSocket`, која је поткласа од `java.net.DatagramSocket`:

```
public class MulticastSocket extends DatagramSocket
```

Као што се може очекивати, `MulticastSocket` се понаша веома слично као `DatagramSocket`: подаци се ставе у `DatagramPacket` објекте који се шаљу и примају помоћу `MulticastSocket`.

За примање података које је multicast-овао удаљени хост, најпре се креира `MulticastSocket` **конструктором `MulticastSocket()`**. Потом се **придружи multicast групи** коришћењем `MulticastSocket` метода **`joinGroup()`**. Овим се сигнализира рутерима на путу између нас и сервера да почну слање података нашим путем и каже локалном хосту да треба да нам прослеђује IP пакете адресиране на multicast групу.

Након што смо се придружили multicast групи, примамо UDP податке као што бисмо и са `DatagramSocket`, тј. **креирамо `DatagramPacket` са низом бајтова који служи као бафер** за податке и **улазимо у петљу у којој примамо податке** позивом метода **`receive()`** наслеђеног од класе `DatagramSocket`. Када више не желимо да примамо податке, **напустимо multicast групу** позивом метода **`leaveGroup()`** сокета. Затим се **сокет** може **затворити** позивом метода **`close()`** наслеђеног од `DatagramSocket`.

Слање података на multicast адресу је слично слању UDP података на unicast адресу.
Није неопходно придружити се multicast групи да би јој се послали подаци.
Креира се `DatagramPacket`, напуни подацима, адресира на multicast групу у пакету и проследи методу `send()`. Разлика је што се мора експлицитно задати вредност TTL пакета.

Постоји један приговор на све ово:
multicast сокети су безбедносна рупа довољно велика да се кроз њу провезе мањи камион.
У складу са тим, неповерљивом кôду који се извршава под контролом SecurityManager-а није допуштено да ради било шта што укључује multicast сокете. Кôду који је remotely loaded уобичајено је допуштено да шаље или прима datagram-е од хоста са ког је download-ован. Међутим, multicast сокети не допуштају да се овакав тип ограничења смести у пакете које шаљу или примају. Након што су подаци послати multicast сокету, имамо врло ограничену и непоуздану контролу над тим који хостови примају те податке. У складу са тим, већина окружења која извршавају удаљени кôд користи конзервативни приступ онемогућавања multicasting-а.

Конструктори

Конструктори су једноставни. Сваки позива еквивалентан конструктор натклase `DatagramSocket`.

`public MulticastSocket() throws SocketException`

Овај конструктор креира сокет везан за анонимни порт (тј. слободан порт који пронађе систем). Користан је за клијенте (тј. програме који иницирају трансфер података) јер они не морају да користе познати порт: прималац одговара на порт садржан у пакету. Ако је неопходно да знате број порта, потражите га методом `getLocalPort()` наслеђеним од `DatagramSocket`. Овај конструктор избацује `SocketException` ако се `Socket` не може креирати. Нпр.

```
try{
      MulticastSocket ms = new MulticastSocket();
      // send some datagrams …
}catch(SocketException se){
      System.err.println(se);
}
```

`public MulticastSocket(int port) throws SocketException`

Овај конструктор креира сокет који прима datagram-е на познатом порту. Аргумент `port` задаје порт на ком овај сокет ослушкује datagram-е. Као са регуларним TCP и UDP unicast сокетима, на Unix систему програм мора бити покренут са root привилегијама како би креирао `MulticastSocket` на порту нумерисаном од 1 до 1023.

Конструктор избацује `SocketException` ако `Socket` не може бити креиран. То се дешава ако немате довољно привилегија да га вежете за порт или је порт за који покушавате да га вежете већ у употреби. Приметите да, како је multicast сокет datagram сокет, што се тиче оперативног система, `MulticastSocket` не може заузети порт који је већ заузео `DatagramSocket` и обрнуто. Нпр. овај фрагмент кôда отвара multicast сокет на порту 4000:

```
 try{
      MulticastSocket ms = new MulticastSocket(4000);
      // receive incoming datagrams …
}catch(SocketException se){
      System.err.println(se);
}
```

`public MulticastSocket(SocketAddress bindAddress) throws IOException`

Почев од Java 1.4 могуће је креирати `MulticastSocket` користећи `SocketAddress` објекат. Ако је `SocketAddress` везан за порт, онда је ово прилично исто као и претходни конструктор. Нпр. овај фрагмент кôда такође отвара multicast сокет на порту 4000:

```
try{
      SocketAddress address = new InetSocketAddress(4000);
      MulticastSocket ms = new MulticastSocket(address);
      // receive incoming datagrams ...
}catch(IOException ex){
      System.err.println(ex);
}
```

Међутим, `SocketAddress` такође може бити везан за одређени мрежни интерфејс на локалном хосту, пре него да ослушкује на свим мрежним интерфејсима. Нпр. овај фрагмент кôда такође отвара multicast сокет на порту 4000 који само ослушкује пакете који пристижу на 192.168.254.32:

```
try{
   SocketAddress address = new InetSocketAddress("192.168.254.32", 4000);
   MulticastSocket ms = new MulticastSocket(address);
   // receive incoming datagrams …
}catch(IOException ex){
   System.err.println(ex);
}
```

Коначно, може се проследити `null` овом конструктору како би се креирао невезани сокет, који се касније може конектовати методом `bind()`. Ово је корисно за подешавање опција сокета које се могу поставити само пре него што се сокет веже за порт. Нпр. овај фрагмент кôда креира multicast

сокет за који је опција SO_REUSEADDR онемогућена (та опција је подразумевано омогућена за multicast сокете):

```
try{
      MulticastSocket ms = new MulticastSocket(null);
      ms.setReuseAddress(false);
      SocketAddress address = new InetSocketAddress(4000);
      ms.bind(address);
      // receive incoming datagrams …
}catch(IOException ex){
      System.err.println(ex);
}
```

Комуникација са multicast групом

Након што је креиран, MulticastSocket може вршити 4 кључне операције:

1. придружити се multicast групи
2. послати податке члановима групе
3. примити податке од групе
4. напустити multicast групу

Класа MulticastSocket поседује методе за операције 1, 2 и 4. Није неопходан нови метод за примање података. Метод receive() суперкласе DatagramSocket одговарајући је за овај задатак. Ове операције могу се вршити произвољним редом, са изузетком да се најпре мора приступити групи пре но што се подаци могу примити од групе или се напустити група.
Није неопходно придружити се групи како би јој се послали подаци и слање и примање података могу бити слободно испреплетани.

```
public void joinGroup(InetAddress address) throws IOException
```

За примање података од MulticastSocket, најпре се мора придружити multicast групи. За придруживање групи, InetAddress објекат за multicast групу проследи се методу joinGroup(). Ако смо се успешно придружили групи, примаћемо све datagram-е намењене тој групи. Након што смо се придружили multicast групи, примамо datagram-е управо као што примамо unicast datagram-е, тј. подесимо DatagramPacket као бафер и проследимо га методу receive() сокета. Нпр.

```
try{
  MulticastSocket ms = new MulticastSocket(4000);
  InetAddress ia = InetAddress.getByName("224.2.2.2");
  ms.joinGroup(ia);
  byte[] buffer = new byte[8192];
  while(true){
        DatagramPacket dp = new DatagramPacket(buffer, buffer.length);
        ms.receive(dp);
        String s = new String(dp.getData(),"8859_1");
        System.out.println(s);
  }
}catch(IOException ex){
  System.err.println(ex);
}
```

Ако адреса којој покушавамо да се придружимо није multicast адреса, тј. није између 224.0.0.0 и 239.255.255.255, метод `joinGroup()` избацује `IOException`.

Један `MulticastSocket` може се придружити већем броју multicast група. Информације о чланству у multicast групама смештене су у multicast рутерима, не у објекту. У овом случају, користимо адресу смештену у долазећем datagram-у да одредимо којој адреси је пакет намењен.

Већи број multicast сокета на истој машини, чак и у истом Јава програму може се придружити истој групи. Ако је тако, сви ће они примати све податке адресиране на ту групу који стижу на локални хост.

```
public void joinGroup(SocketAddress address, NetworkInterface interface)
                                                        throws IOException
```

Java 1.4 додаје ову варијанту метода `joinGroup()` која омогућује придруживање multicast групи само на задатом локалном мрежном интерфејсу. Рецимо, прокси сервер или firewall могу то користити да задају да ће прихватати multicast податке од интерфејса повезаног на LAN, али не од интерфејса повезаног на глобални Интернет.

Нпр. овај фрагмент кôда покушава да се придружи групи са IP адресом 224.2.2.2 на мрежном интерфејсу са именом "eth0", ако такав интерфејс постоји. Ако не постоји такав интерфејс, онда се придружује на свим доступним мрежним интерфејсима:

```
MulticastSocket ms = new MulticastSocket(4000);
SocketAddress group = new InetSocketAddress("224.2.2.2", 40);
NetworkInterface ni = NetworkInterface.getByName("eth0");
if(ni!=null){
      ms.joinGroup(group, ni);
}else{
      ms.joinGroup(group);
}
```

Осим што захтева додатни аргумент који одређује мрежни интерфејс са кога ослушкује, ово се понаша прилично слично као метод `joinGroup()` са једним аргументом. Нпр. прослеђивање објекта `SocketAddress` који не представља multicast групу као првог аргумента доводи до избацивања `IOException`.

```
public void leaveGroup(InetAddress address) throws IOException
```

Метод `leaveGroup()` сигнализира да више не желимо да примамо datagram-е од задате multicast групе. Сигнал се шаље одговарајућем multicast рутеру, казујући му да престане да нам шаље datagram-е. Ако адреса коју покушавамо да напустимо није multicast адреса (тј. није између 224.0.0.0 и 239.255.255.255), метод избацује `IOException`. Међутим, нема изузетка ако напуштамо multicast групу којој нисмо придружени.

public void leaveGroup(SocketAddress multicastAddress, NetworkInterface interface) throws IOException

Java 1.4 такође омогућује да задамо да више не желимо да примамо datagram-е на једном одређеном мрежном интерфејсу. Можда желимо да наставимо да примамо datagram-е на другим мрежним интерфејсима. Метод је вероватно укључен углавном ради симетрије са `joinGroup()`.

public void send(DatagramPacket packet, byte ttl) throws IOException

Слање података помоћу `MulticastSocket` слично је као слање података помоћу `DatagramSocket`. Напунимо `DatagramPacket` објекат подацима и пошаљемо га методом `send()` наслеђеним од `DatagramSocket`:

```
public void send(DatagramPacket p) throws IOException
```

Подаци се шаљу сваком хосту који припада multicast групи на коју је пакет адресиран. Нпр.

```
try{
  InetAddress ia = InetAddress.getByName("experiment.mcast.net");
  byte[] data = "Here's some multicast data\r\n".getBytes();
  int port = 4000;
  DatagramPacket dp = new DatagramPacket(data, data.length, ia, port);
  MulticastSocket ms = new MulticastSocket();
  ms.send(dp);
}catch(IOException ex){
  System.err.println(ex);
}
```

Међутим, класа `MulticastSocket` додаје преклопљену (overloaded) варијанту метода `send()` која допушта да задамо вредност `ttl` за поље Time-To-Live. **Подразумевано, метод `send()` користи вредност 1 за TTL; тј. пакети не путују изван локалне подмреже.** Међутим, ово подешавање се може променити за појединачни пакет прослеђивањем целог броја од 0 до 255 као другог аргумента методу `send()`. Нпр.

```
DatagramPacket dp = new DatagramPacket(data, data.length, ia, port);
MulticastSocket ms = new MulticastSocket();
ms.send(dp, 64);
```

public void setInterface(InetAddress address) throws SocketException

На хосту који има више мрежних интерфејса, метод `setInterface()` бира мрежни интерфејс који се користи за multicast слање и примање. Метод избацује `SocketException` ако аргумент `InetAddress` не представља адресу мрежног интерфејса на локалној машини. Нејасно је због чега се мрежни интерфејс непроменљиво поставља у конструкторима за unicast `Socket` и `DatagramSocket` објекте, али је променљив и поставља се посебним методом за `MulticastSocket` објекте. Ради сигурности, поставите интерфејс непосредно након конструисања `MulticastSocket` и не мењајте га након тога. Овако би се могао користити метод `setInterface()`:

```
MulticastSocket ms;
InetAddress ia;
```

```
try{
      ia = InetAddress.getByName("www.ibiblio.org");
      ms = new MulticastSocket(2048);
      ms.setInterface(ia);
      // send and receive data …
}catch(UnknownHostException ue){
      System.err.println(ue);
}catch(SocketException se){
      System.err.println(se);
}
```

public InetAddress getInterface() throws IOException

Ако је потребно да знамо адресу интерфејса за који је сокет везан, зовемо `getInterface()`. Није јасно зашто би овај метод избацио изузетак; у сваком случају, морамо бити спремни за то. Нпр.

```
try{
      MulticastSocket ms = new MulticastSocket(2048);
      InetAddress ia = ms.getInterface();
}catch(IOException ex){
      System.err.println(ex);
}
```

public void setNetworkInterface(NetworkInterface interface)
throws SocketException

Метод `setNetworkInterface()` има исту сврху као и метод `setInterface()`; тј. бира мрежни интерфејс који се користи за multicast слање и примање. Међутим, он то ради на основу имена мрежног интерфејса, попут "eth0" (енкапсулираног у `NetworkInterface` објекат) пре него на IP адреси везаној за тај мрежни интерфејс (енкапсулираној у `InetAddress` објекту). Метод избацује `SocketException` ако `NetworkInterface` прослеђен као аргумент не представља мрежни интерфејс на локалној машини.

public NetworkInterface getNetworkInterface() throws SocketException

Метод `getNetworkInterface()` враћа `NetworkInterface` објекат који представља мрежни интерфејс на коме текући `MulticastSocket` ослушкује податке. Ако ниједан мрежни интерфејс није експлицитно постављен у конструктору нити методом `setNetworkInterface()`, враћа "placeholder" објекат са адресом "0.0.0.0" и индексом -1. Нпр. овај фрагмент кôда штампа мрежни интерфејс који сокет користи:

```
NetworkInterface intf = ms.getNetworkInterface();
System.out.println(intf.getName());
```

public void setTimeToLive(int ttl) throws IOException

Метод `setTimeToLive()` враћа подразумевану TTL вредност која се користи за пакете послате сокетом користећи метод `send(DatagramPacket dp)` наслеђен од `DatagramSocket`

(наспрам методу `send(DatagramPacket dp, byte ttl)` класе `MulticastSocket`). Овај метод доступан је почев од Java 1.2. У Java 1.1 мора се уместо њега користити метод `setTTL()`:

```
public void setTTL(byte ttl) throws IOException
```

Метод `setTTL()` је почев од Java 2 застарео јер допушта само TTL вредностиод 1 до 127 уместо читавог опсега од 1 до 255.

`public int getTimeToLive() throws IOException`

Метод `getTimeToLive()` враћа подразумевану TTL вредност за MulticastSocket. Није много потребан. Такође је доступан почев од Java 1.2. У Java 1.1 уместо њега се мора користити метод `getTTL()`:

```
public byte getTTL() throws IOException
```

Метод `getTTL()` је застарео почев од Java 1.2 јер не рукује исправно TTL вредностима већим од 127 - одсеца их на 127. Метод `getTimeToLive()` може руковати читавим опсегом од 1 до 255 без одсецања јер враћа `int` уместо `byte`.

`public void setLoopbackMode(boolean disable) throws SocketException`

Да ли ће хост примати или не multicast пакете које је он сâм послао зависи од платформе. Прослеђивање `true` методу `setLoopbackMode()` указује да не желимо да примамо пакете које шаљемо. Прослеђивање `false` указује да желимо. Међутим, то је само препорука. Имплементације нису дужне да раде како захтевамо.

`public boolean getLoopbackMode() throws SocketException`

Како је loopback мôд само препорука која не мора бити испоштована на свим системима, важно је проверити који је loopback мôд ако и шаљемо и примамо пакете. Метод `getLoopbackMode()` враћа `true` ако се послати пакети не примају, а `false` ако се примају. (Харолду, а и мени, ово делује наопако. Харолд сумња да је метод написао програмер пратећи болесну конвенцију да подразумевана вредност увек треба да буде `true`.)

Ако систем прима назад пакете, а не желимо то, неопходно је да некако препознамо те пакете и одбацимо их. Ако систем не прима пакете назад, а желимо то, можемо да чувамо копије пакета које смо послали и ручно их убацимо у своју интерну структуру података у исто време када их пошаљемо. Можемо да замолимо за понашање које желимо методом `setLoopback()`, али не можемо рачунати да ће то бити уважено.

Два једноставна примера

Већина multicast сервера не бира са ким разговара. Према томе, једноставно је придружити се групи и гледати податке који се шаљу серверу.

Пример 1 је класа `MulticastSniffer` која чита име multicast групе из командне линије, конструише `InetAddress` од тог hostname и креира `MulticastSocket` који покушава да се придружи multicast групи на том hostname.

Ако покушај успе, `MulticastSniffer` прима datagram-е од сокета и штампа њихов садржај на `System.out`. Програм је користан пре свега да верификује да примамо multicast податке на одређеном хосту. Већина multicast података је бинарна и неће бити разумљива када се одштампа као ASCII.

Програм започиње читањем имена и порта multicast групе као аргумената командне линије. Потом креира нови `MulticastSocket ms` на задатом порту. Сокет се придружује multicast групи на задатом `InetAddress`. Затим улази у петљу у којој чека пакете да стигну. Након што сваки пакет пристигне, програм чита његове податке, конвертује их у ISO Latin-1 `String` и штампа на `System.out`. Коначно, када корисник прекине програм или се избаци изузетак, сокет напушта групу и затвара се.

Пример 2 је класа `MulticastSender` која шаље податке multicast групи. Прилично просто, све.

Пример чита адресу multicast групе, број порта и опциони TTL из командне линије. Затим у низ бајтова `data` уписује садржај стринга "Here's some multicast data\r\n", користећи метод `getBytes()` класе `java.lang.String` и смешта овај низ у `DatagramPacket dp`. Потом конструише `MulticastSocket ms`, који се придружује групи `ia`. Након што се придружио групи, `ms` шаље datagram пакет `dp` групи `ia` 10 пута. Вредност TTL је постављена на 1 како бисмо се осигурали да подаци не излазе ван локалне подмреже. Након што је послао податке, ms напушта групу и затвара се.

Покрените `MulticastSniffer` на једној машини Ваше локалне подмреже. Ослушкујте групу all-systems.mcast.net на порту 4000, овако:

```
>> java MulticastSniffer all-systems.mcast.net 4000
```

Затим пошаљите податке тој групи извршавањем `MulticastSender` на другој машини своје локалне подмреже. Можете га такође покренути у другом прозору на истој машини, иако та опција није толико узбудљива. Међутим, морате започети извршавање `MulticastSniffer` пре него `MulticastSender`. За слање групи all-systems.mcast.net на порту 4000 покретање програма `MulticastSender` изгледа овако:

```
>> java MulticastSender all-systems.mcast.net 4000
```

На првој машини треба да видите овакав излаз:

```
Here's some multicast data

Here's some multicast data

Here's some multicast data

Here's some multicast data

Here's some multicast data

Here's some multicast data

Here's some multicast data

Here's some multicast data

Here's some multicast data

Here's some multicast data
```