Komunikace typu multicast

Jiří Hýsek, xhysek02

8. listopadu 2004

Obsah

1 Úvod 3

2 Multicastové skupiy 3

3 Směrování – význam TTL v multicast datagramech 3

3.1 Implementace . . . 4

4 Odesílání multicast datagramu 4

4.1 Vytvoření socketu (schránky) . . . 4

4.2 Odeslání dat . . . 5

4.2.1 Funkce sendto . . . 5

4.2.2 Ukázka použití sendto, pro odesílání dat do multicastové skupiny . . . 5

5 Příjem multicast datagramu 6

5.1 Připojení k multicastové skupině . . . 6

5.2 Odpojení z multicastové skupiny . . . 6

5.3 Příjem dat . . . 6

5.3.1 Čekání na příchozí data . . . 6

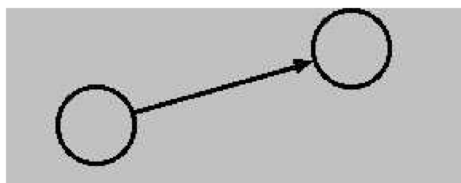
5.3.2 Funkce recvfrom . . . 7

5.3.3 Ukázka použití recvfrom pro přečtení dat zaslaných do multicastové skupiny . . . 7

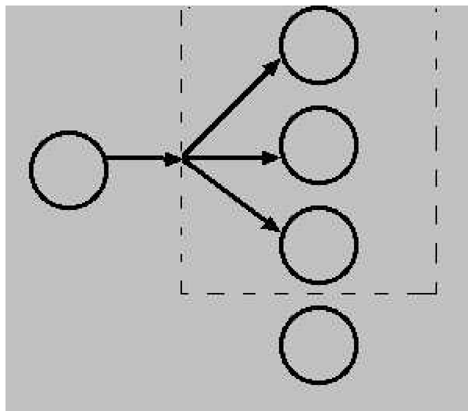
6 Závěr 7

1 Úvod

Běžná TCP spojení jsou spojení typu *unicast*. Jde tedy o komunikaci mezi dvěmi procesy běžících na různých nebo i stejných uzlech sítě. Pokud tedy je požadováno stejná data poslat více příjemcům, nezbude nic jiného, než data poslat pro každého příjemce znovu. To je samozřejmě velmi neefektivní. Tento problém řeší komunikace typu *multicast*. Příjemci jsou organizováni do *multicastových skupin*. Pokud odešleme paket do této skupiny, obrdží jej všichni její členové. Protože je TCP unicast spojení, multicast se realizuje pomocí UDP. Z toho vyplývá, že multicast nekontroluje správné pořadí paketů, ani to, zda vůbec byl paket řádně doručen.



Obr. 1 – komunikace typu unicast



Obr.2 – komunikace typu multicast

2 Multicastové skupiy

Každá skupina má svou adresu. Adresa skupiny vypadá stejně jako IP adresa. Pro adresy multicastových skupin jsou vyčleněny IP adresy třídy D – tedy rozsah 224.0.0.0 až 239.255.255.255. Rozsah adres od 224.0.0.0 do 224.0.0.255 je rezervován pro potřeby multicastového směrování a tyto adresy by neměly být používány aplikačními programy. Datagramy adresované do těchto skupin, nebudou multicastovým směrovačem předány dál. Existuje několik speciálních adres, jako např. 224.0.0.1, která zastupuje adresu všech skupin, je to tedy adresa všech klientů na síti, kteří jsou schopni komunikovat pomocí multicast.

3 Směrování – význam TTL v multicast datagramech

TTL (time to live) v IP určuje dobu života datagramu, tedy maximální počet průchodů přes směrovač. V multicast komunikaci se využívá pro omezování vzdálenosti, kam až datagram může dorazit, ovšem trochu odlišným způsobem. Nestačí, aby směrovač TTL pouze snížil a datagram zahodil při dosažení TTL = 0. Jsou definovány tyto hodnoty TTL pro daná omezení:

- TTL = 0 – omezeno na stejný počítač
- TTL = 1 – omezeno na stejnou podsíť
- TTL = 32 – omezeno na stejnou síť

- TTL = 64 – stejný region
- TTL = 128 – stejný kontinent
- TTL = 255 – neomezeno

Omezení nefungují jen tak sama od sebe, směrovač musí být správně nakonfigurován. Směrovač musí mít u každého rozhraní definovanou hodnotu TTL. Datagramy s vyšším nebo stejným TTL pošle dál a naopak datagramy s nižší hodnotou TTL zahazuje. TTL u komunikace typu multicast funguje jako "treshold".

3.1 Implementace

Implicitní hodnota TTL pro multicastové pakety je 1, což znamená, že jsou omezeny pouze pro jednu lokální síť. Můžeme ji však nastavit pomocí funkce **setsockopt**:

```
unsigned char ttl = 32;
setsockopt(sock, IPPROTO_IP, IP_MULTICAST_TTL, &ttl, sizeof(ttl));
```

První parametrem je socket, přes který komunikace probíhá. Druhý parametr je úroveň, na které se soketem manipulujeme. V našem případě provádíme změny na úrovni IP protokolu. Další parametr je druh volby, kterou měníme. Dalším parametrem je konkrétní nastavovaná hodnota. Její typ je závislý na nastavované volbě, proto musíme zadat ukazatel na ni a v posledním parametru délku.

4 Odesílání multicast datagramu

4.1 Vytvoření socketu (schránky)

Dříve než si popíšeme odesílání a příjem dat, ukážeme si jak vytvořit socket pro multicastovou komunikaci. Socket vytvoříme tímto způsobem nezávisle na tom, zda chceme data odesílat nebo přijímat.

Jak již bylo řečeno v úvodu, pro multicast datagramy využíváme protokolu UDP. Socket tedy vytvoříme voláním funkce **socket**:

```
int sockfd;
sockfd = socket(PF_INET, SOCK_DGRAM, 0);
```

Tento kód přiřadí do proměnné sockfd desktriptor na vytvořený socket. První parametr určuje **rodinu** (doménu) protokolů, které budeme používat. Kromě PF_INET to mohou být například PF_INET6 pro IPv6, PF_UNIX pro lokální komunikaci (na jednom počítači), a další[1].

Druhý parametr specifikuje sémantiku komunikace, nebo **typ** protokolu. Např. SOCK_STREAM znamená navazované spolehlivé spojení (jako např. TCP), my však použijeme SOCK_DGRAM, což znamená nespojovanou komunikaci bez zajištění správného doručení paketů (UDP). Samozřejmě i zde existuje více voleb[1].

Poslední parametr specifikuje komunikační **protokol**. Číslo je závislé na předchozích parametrech, pokud tedy zadáme PF_INET a SOCK_DGRAM, protokol číslo 0 znamená UDP. Pokud bychom jako druhý parametr zvolili SOCK_STREAM, protokol číslo 0 by znamenal TCP. 0 tedy znamená výchozí protokol pro danou rodinu a typ protokolu.

[1] Všechny volby naleznete v manuálových stránkách příkazu socket.

4.2 Odeslání dat

4.2.1 Funkce sendto

Pro odeslání dat slouží funkce **sendto**. V manuálových stránkách [4] najdeme její deklaraci:

```
ssize_t sendto(int s, const void *msg, size_t len,
int flags, const struct sockaddr *to, socklen_t tolen);
```

Prvním parametrem *s* je deskriptor socketu, který jsme si vytvořili voláním funkce socket().

Druhý parametr *msg* je ukazatel na odesílaná data, další parametr *len* je délka těchto dat.

Parametrem *flags* se definují další volby. My žádné používat nebudeme, tudíž vás odkážu pouze na [4].

Parametr *to* obsahuje informace o příjemci těchto dat. Tady trochu odbočím od problematiky multicast komunikace a obajsním způsob, jakým se obecně v inetrnetových aplikacích zadávají informace o počítači. Informace se předávají v podobě struktury *sockaddr*, která vypadá následovně:

```
struct sockaddr {
  u_char sa_len;
  sa_family_t sa_family;
  char sa_data[14];
}
```

Obsahuje délku struktury, rodinu protokolů (AF_INET například) a nějaká data. V těchto datech je uložen port a adresa. V praxi, jde-li o rodinu protokolů AF_INET, se informace o příjemci zapisují do struktury sockaddr_in a ta se potom pouze přetypuje na sockaddr. sockaddr_in vypadá takto:

```
struct sockaddr_in {
  u_char sin_len;
  u_char sin_family;
  u_short sin_port;
  struct in_addr sin_addr;
  char sin_zero[8];
}
```

Oproti sockaddr tu máme 3 parametry navíc. *sin_port* nese informaci o portu, na kterém bude komunikace probíhat. Bity jsou však uvedeny v opačném pořadí. Tedy MSB (most significant bit) je posílán jako první. Proto existuje převodní funkce **htons**.

Další prvek je struktura, která se uchovává pouze z historických důvodů. Obsahuje totiž pouze 1 prvek *in_addr_t s_addr*. Obsahuje IP adresu, jejíž bity jsou opět v opačném pořadí. Naštěstí existuje i zde předvodní funkce **inet_addr**, která převádí textovou reprezentaci IP adresy na požadovaný formát.

Poslední *sin_zero*, je nevyužito, je to pouze výplň, aby byla délka struktury shodná s délkou struktury sockaddr.

4.2.2 Ukázka použití sendto, pro odesílání dat do multicastové skupiny

Předpokládáme, že již máme vytvořený socket, deskriptor je uložen v proměnné *sockfd*. Nejdříve vyplníme informace o příjemci:

```
struct sockaddr_in addr;

addr.sin_family = AF_INET;
addr.sin_addr.s_addr = inet_addr("225.0.0.50");
addr.sin_port = htons(1234);
```

A zavoláme sendto:

```
sendto(sockfd, "hello", 5, 0, (struct sockaddr *)&addr, sizeof(addr));
```

Tímto jsme poslali řetězec "hello" do skupiny 225.0.0.50 programům poslouchající na portu 1234.

5 Příjem multicast datagramu

5.1 Připojení k multicastové skupině

Aby byla aplikace schopna přijímat multicast datagramy, musí být daný socket členem nějaké multicastové skupiny. To se zařídí funkcí **setsockopt**.

```
struct ip_mreq mreq;

mreq.imr_multiaddr.s_addr = inet_addr("225.0.0.50");
mreq.imr_interface.s_addr = htonl(INADDR_ANY);

setsockopt(sock,IPPROTO_IP,IP_ADD_MEMBERSHIP,&mreq,sizeof(mreq));
```

Struktura ip_mreq obsahuje informace o skupině, ke které se připojujeme. *imr_multiaddr* obsahuje adresu multicastové skupiny. *imr_interface* je adresa rozhraní, ke kterému se připojíme. Zde můžeme použít *INADDR_ANY*, což znamená, že přijímáme multicastové datagramy z jakéhokoli rozhraní.

Jeden socket je možné připojit do více skupin.

5.2 Odpojení z multicastové skupiny

Členství v dané skupině zrušíme analogicky, ovšem jako třetí parametr zadáme *IP_DROP_MEMBERSHIP*.

5.3 Příjem dat

5.3.1 Čekání na příchozí data

Pouhé připojení k multicastové skupině samozřejmě nestačí. Jako u všech serverových internetových aplikací, musí program čekat na určitém portu na příchozí spojení, v našem případě pouze data (jde o nespojovanou službu). A to zařídíme příkazem **bind**:

```
struct sockaddr_in addr;

addr.sin_family = AF_INET;
```

```
addr.sin_addr.s_addr = htonl(INADDR_ANY);
addr.sin_port = htons(1234);

bind(sockfd,(struct sockaddr *)&addr, sizeof(addr));
```

Po provedení těchto řádků kódu je program připraven číst data, která jsou posílána na port 1234.

5.3.2 Funkce recvfrom

Pro přijetí datagramu použijeme funkci **recvfrom**.

```
ssize_t recvfrom(int s, void *buf, size_t len, int flags,
struct sockaddr *from, socklen_t *fromlen);
```

První parametr je deskriptor socketu, z nějž budeme data číst. Druhý parametr je buffer, do nějž se přečtená data uloží. Třetí parametr *len* je maximální délka čtených dat. Dále máme možnost opět zadat různé modifikující flagy a nakonec *from*, což je ukazatel na strukturu *sockaddr*, kam se uloží informace o odesílateli. Z 4.2.1 je zřejmé, že zde lze použít i ukazatel na strukturu *sockaddr_in*. Poslední parametrem se předává délka této struktury.

Funkce vrací počet bajtů, které se jí podařilo přečíst.

5.3.3 Ukázka použití recvfrom pro přečtení dat zaslaných do multicastové skupiny

Předpokládejme, že máme vytvořen socket, který je již v multicastové skupině, a jeho deskriptor je uložen v proměnné *sockfd*.

```
int ret;
int addr_len;
sockaddr_in addr;
char buf[1024];

addr_len = sizeof(addr);
while ((ret = recvfrom(sockfd, buf, 1024, 0,
                        (struct sockaddr *)&addr, &addr_len))) {
        printf("Prijato %dB: '%s'\n", addr_len, buf);
}
```

Po přijetí dat se do struktury *addr* uloží informace o odesílateli dat.

6 Závěr

Hlavní výhodou a zároveň důvodem pro vznik a použití multicastové komunikace je výrazné snížení množství přenášených dat. Samozřejmě to platí pouze pro specifická použití. A to tam, kde jsou stejná data rozesílána více příjemcům a případné ztráty dat nejsou nijak závažné. To je například případ videokonferencí, streamování audia či videa v reálném čase a podobně. V oblasti multicast přenosu se věci stále hýbou vpřed. Vyvíjí se protokoly pro spolehlivý multicast (např. MTP – multicast transport protocol nebo URGC – uniform reliable group communication), nebo protokoly na multicastu založené, jako třeba MFTP (multicast file transfer protokol).

Použitá literatura

[1] de Goyeneche, J.: "Multicast over TCP/IP HOWTO" [elektronický dokument], 1998.
Dostupný na URL: http://www.tdlp.org/HOWTO/Multicast-HOWTO.html
[2]An Introduction to IP Multicast [elektronický dokument].
Dostupný na URL: http://ntrg.cs.tcd.ie/undergrad/4ba2/transport/
[3] manuálová stránka funkce socket [elektronický dokument], dostupný jako součást UNIX-like systémů
[4] manuálová stránka funkce sendto [elektronický dokument], dostupný jako součást UNIX-like systémů
[5] manuálová stránka funkce recvfrom [elektronický dokument], dostupný jako součást UNIX-like systémů

Příloha

Zdrojový kód jednoduchého příjemce multicast datagramů:

```
#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <arpa/inet.h>
#include <stdio.h>
#include <string.h>
#include <unistd.h>
#include <stdlib.h>

#define MULTICAST_PORT  12345
#define MULTICAST_GROUP "225.0.0.38"

int main(void) {
     struct sockaddr_in addr;
     int sock;
     int addrlen;
     struct ip_mreq mreq;
     char buf[256] = {0};
     u_int yes = 1;

     /* vytvorime socket, deskriptor ulozime do promenne sock */
     if ((sock = socket(AF_INET, SOCK_DGRAM, 0)) == -1) {
            perror("socket");
            exit(1);
     }

     /* povolime pripojeni vice socketu k jednomu portu; pro testovaci ucely */
     if (setsockopt(sock ,SOL_SOCKET, SO_REUSEADDR, &yes, sizeof(yes)) < 0) {
           perror("Reusing ADDR failed");
           exit(1);
     }

     /* nastavime adresu a port, na kterem program posloucha */
     memset(&addr, 0, sizeof(addr));
     addr.sin_family = AF_INET;
     addr.sin_addr.s_addr = htonl(INADDR_ANY);
     addr.sin_port = htons(MULTICAST_PORT);


     /* program se "povesi" na dany port */
     if (bind(sock, (struct sockaddr *)&addr, sizeof(addr)) == -1) {
```

```
            perror("bind");
            close(sock);
            exit(1);
    }

    /* definice multicastove skupiny */
    mreq.imr_multiaddr.s_addr = inet_addr(MULTICAST_GROUP);
    mreq.imr_interface.s_addr = htonl(INADDR_ANY);

    /* pridani socketu do multicastove skupiny */
    if (setsockopt(sock, IPPROTO_IP, IP_ADD_MEMBERSHIP,
        &mreq, sizeof(mreq)) == -1)
   {
           perror("setsockopt");
           close(sock);
           exit(1);
    }

    addrlen = sizeof(addr);
    /* dokud mame co cist, cteme a vypisujeme na obrazovku */
    while (recvfrom(sock, buf, sizeof(buf), 0, (struct sockaddr *)&addr,
           &addrlen) > 0)
    {
           printf("%s", buf);
           memset(buf, sizeof(buf), 0);
    }
    return 0;
}
```

Zdrojový kód jednoduchého odesílatele multicast datagramů:

```
#include <sys/socket.h>
#include <sys/types.h>
#include <netinet/in.h>
#include <arpa/inet.h>
#include <stdio.h>
#include <string.h>
#include <unistd.h>
#include <stdlib.h>

#define MULTICAST_PORT  12345
#define MULTICAST_GROUP "225.0.0.38"

int main() {
     struct sockaddr_in addr;
     int sock;
```

```
    char buf[256] = {0};

    /* vytvorime socket, deskriptor ulozime do promenne sock */
    if ((sock = socket(AF_INET,SOCK_DGRAM,0)) == -1) {
        perror("socket");
        exit(1);
    }

    /* nastavime adresu a port skupiny, kam budou data odeslana */
    memset(&addr, 0, sizeof(addr));
    addr.sin_family = AF_INET;
    addr.sin_addr.s_addr = htonl(INADDR_ANY);
    addr.sin_port = htons(MULTICAST_PORT);

    /* cteme z se stdin a posilame pomoci sendto */
    while (fgets(buf, sizeof(buf), stdin)) {
        if (sendto(sock, buf, strlen(buf), 0,
           (struct sockaddr *) &addr, sizeof(addr)) == -1)
        {
            perror("sendto");
            close(sock);
            exit(1);
        }
        memset(buf, sizeof(buf), 0);
    }
    return 0;
}
```