Socket Multicast

Teoria ed implementazione

Roberto Marino, PhD

Università di Messina
CdL in Informatica — Reti e Sistemi Distribuiti Mod.B

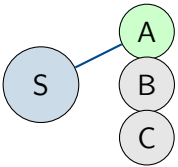
March 26, 2026



UniMe
1548

Tre modi di consegnare un pacchetto

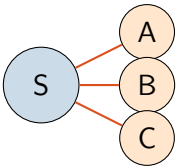
Unicast



1 sorgente → 1 dest.

`SOCK_DGRAM` standard

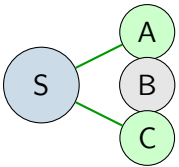
Broadcast



1 sorgente → **tutti**

255.255.255.255

Multicast



1 sorgente → **gruppo**

224.0.0.0/4

Multicast è la via di mezzo: consegna efficiente a un *sottoinsieme* di host che si sono esplicitamente **iscritti** al gruppo.

Perché non usare semplicemente broadcast?

- Il broadcast raggiunge **ogni** host sulla LAN — compresi quelli che non sono interessati
- Ogni host deve processare il pacchetto fino al livello applicazione per scoprire che non lo riguarda → spreco di CPU
- I router **non instradano** il broadcast fuori dalla subnet locale
- Il broadcast non esiste in IPv6 (sostituito interamente da multicast)

Multicast risolve tutti questi problemi:

- Solo gli host iscritti ricevono i pacchetti
- Funziona su scala internet (se la rete lo supporta)

Il blocco 224.0.0.0/4

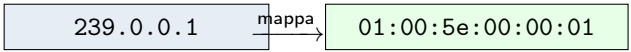
Range	Scope	Esempio d'uso
224.0.0.0 – 224.0.0.255	Link-local	OSPF, RIP, mDNS
224.0.1.0 – 238.255.255.255	Global	Streaming, videoconf
239.0.0.0 – 239.255.255.255	Administratively scoped	LAN privata

Per i nostr esperimenti useremo il range **239.x.x.x** (administratively scoped): i pacchetti non usciranno mai dalla rete locale, come le subnet private 192.168.x.x per unicast.

Nell'HO userai: `239.0.0.1` porta `5000`

MAC multicast — come funziona a livello 2

- Ogni indirizzo IP multicast ha un corrispondente **MAC multicast**
- Formula: `01:00:5e:XX:XX:XX` dove XX:XX:XX derivano dagli ultimi 23 bit dell'IP
- Esempio: `239.0.0.1` → `01:00:5e:00:00:01`



Come un host si iscrive a un gruppo

- **IGMP** è il protocollo tra sender e receiver multicast
- Quando chiamimo `setsockopt(IP_ADD_MEMBERSHIP)` il kernel manda un **IGMP Join**
- Quando chiudi il socket (o `IP_DROP_MEMBERSHIP`) il kernel manda un **IGMP Leave**
- Il tutto è **completamente trasparente**: lo fa il kernel, non la tua app

setsockopt() — prototipo e parametri

```
#include <sys/socket.h>

int setsockopt(int sockfd,          /* il socket da configurare        */
               int level,           /* a quale livello si applica      */
               int optname,         /* quale opzione vogliamo cambiare */
               const void *optval,  /* valore da impostare             */
               socklen_t optlen);   /* dimensione di optval            */
/* Ritorna 0 successo, -1 errore (controlla errno) */
```

level dice al kernel a quale strato dello stack appartiene l'opzione:

level	Strato	Esempio opzione
SOL_SOCKET	Socket generico	SO_REUSEADDR, SO_RCVBUF
IPPROTO_IP	IP (livello 3)	IP_TTL, IP_ADD_MEMBERSHIP
IPPROTO_TCP	TCP (livello 4)	TCP_NODELAY

Errore comune: usare SOL_SOCKET invece di IPPROTO_IP per le opzioni multicast. Il kernel in alcuni casi non restituisce errore — ignora silenziosamente l'opzione.

setsockopt() — le quattro opzioni multicast

```
/* 1. SOL_SOCKET     piu' processi in bind sulla stessa porta        */
int yes = 1;
setsockopt(fd, SOL_SOCKET, SO_REUSEADDR, &yes, sizeof(yes));

/* 2. IPPROTO_IP     limita la portata del pacchetto (1 = LAN only) */
int ttl = 1;
setsockopt(fd, IPPROTO_IP, IP_MULTICAST_TTL, &ttl, sizeof(ttl));

/* 3. IPPROTO_IP     iscrizione al gruppo (il kernel manda IGMP Join)
      optval e' una struct, non un int: il tipo di optval cambia
      a seconda dell'opzione     passa sempre sizeof() corretto.     */
struct ip_mreq mreq;
inet_pton(AF_INET, "239.0.0.1", &mreq.imr_multiaddr);
mreq.imr_interface.s_addr = htonl(INADDR_ANY);
setsockopt(fd, IPPROTO_IP, IP_ADD_MEMBERSHIP, &mreq, sizeof(mreq));

/* 4. IPPROTO_IP     il sender riceve i propri pacchetti? (default=1) */
int loop = 0;
setsockopt(fd, IPPROTO_IP, IP_MULTICAST_LOOP, &loop, sizeof(loop));
```

`IP_ADD_MEMBERSHIP` è l'unica che alloca stato nel kernel e genera traffico di rete (IGMP Join). Le altre tre modificano solo parametri locali al socket.

Creare il socket: receiver (parte 1)

Listing 1: receiver.c — creazione socket e binding

```
/* UDP socket     multicast usa sempre UDP */
int sockfd = socket(AF_INET, SOCK_DGRAM, 0);

/* SO_REUSEADDR: permette a piu' processi di bind sulla stessa porta
   (utile per test con piu' receiver sulla stessa macchina)      */
int yes = 1;
setsockopt(sockfd, SOL_SOCKET, SO_REUSEADDR, &yes, sizeof(yes));

/* bind sull'indirizzo multicast e sulla porta del gruppo.
   Alcuni testi usano INADDR_ANY; usare l'indirizzo multicast
   e' piu' preciso: filtra a livello kernel i pacchetti
   diretti ad altri gruppi sulla stessa porta.                   */
struct sockaddr_in addr = {
    .sin_family      = AF_INET,
    .sin_port        = htons(MCAST_PORT),     /* 5000 */
    .sin_addr.s_addr = inet_addr(MCAST_GROUP) /* 239.0.0.1 */
};
bind(sockfd, (struct sockaddr *)&addr, sizeof(addr));
```

Iscriversi al gruppo: IP_ADD_MEMBERSHIP (parte 2)

Listing 2: receiver.c — iscrizione al gruppo multicast

```
/*
 * struct ip_mreq descrive l'iscrizione:
 *   imr_multiaddr: indirizzo IP del gruppo
 *   imr_interface: quale interfaccia usare (INADDR_ANY = default)
 *
 * Questa chiamata fa due cose:
 *   1. Dice al kernel di ricevere pacchetti per questo gruppo
 *   2. Il kernel manda un IGMP Join al router locale
 */

struct ip_mreq mreq;
inet_pton(AF_INET, MCAST_GROUP, &mreq.imr_multiaddr);
mreq.imr_interface.s_addr = htonl(INADDR_ANY);

setsockopt(sockfd,
           IPPROTO_IP,        /* livello IP */
           IP_ADD_MEMBERSHIP, /* unisciti al gruppo */
           &mreq, sizeof(mreq));

/* Da questo momento recv() riceve i datagrammi del gruppo */
```

Il sender: nessuna iscrizione richiesta

Listing 3: sender.c — invio a un gruppo multicast

```
/*
 * Il sender non deve iscriversi al gruppo.
 * Basta un normale socket UDP e spedire all'indirizzo del gruppo.
 * Il kernel si occupa di tutto: incapsula in UDP/IP e usa
 * il MAC multicast derivato dall'IP del gruppo.
 */
int sockfd = socket(AF_INET, SOCK_DGRAM, 0);

/* IP_MULTICAST_TTL: quanti router puo' attraversare il pacchetto.
   1 = link-local (non esce dalla LAN). Default = 1.            */
int ttl = 1;
setsockopt(sockfd, IPPROTO_IP, IP_MULTICAST_TTL, &ttl, sizeof(ttl));

struct sockaddr_in dest = {
    .sin_family      = AF_INET,
    .sin_port        = htons(MCAST_PORT),
    .sin_addr.s_addr = inet_addr(MCAST_GROUP)
};

sendto(sockfd, msg, strlen(msg), 0,
       (struct sockaddr *)&dest, sizeof(dest));
```

Ricezione e uscita dal gruppo

Listing 4: receiver.c — ricezione e cleanup

```
char buf[1024];
struct sockaddr_in sender_addr;
socklen_t addrlen = sizeof(sender_addr);

/* recvfrom riempie sender_addr con IP e porta del sender */
ssize_t n = recvfrom(sockfd, buf, sizeof(buf) - 1, 0,
                     (struct sockaddr *)&sender_addr, &addrlen);
buf[n] = '\0';

char sender_ip[INET_ADDRSTRLEN];
inet_ntop(AF_INET, &sender_addr.sin_addr, sender_ip, sizeof(sender_ip));
printf("[%s] %s\n", sender_ip, buf);

/* --- Cleanup: abbandona il gruppo prima di chiudere --- */
setsockopt(sockfd, IPPROTO_IP, IP_DROP_MEMBERSHIP,
           &mreq, sizeof(mreq));
/* Il kernel mandera' automaticamente un IGMP Leave        */
close(sockfd);
```

Passi per receiver e sender

Receiver

1. `socket(AF_INET, SOCK_DGRAM, 0)`
2. `setsockopt(SO_REUSEADDR)`
3. `bind()` sull'indirizzo gruppo + porta
4. `setsockopt(IP_ADD_MEMBERSHIP)`
   → kernel invia IGMP Join
5. `recvfrom()` loop
6. `setsockopt(IP_DROP_MEMBERSHIP)`
7. `close()`

Sender

1. `socket(AF_INET, SOCK_DGRAM, 0)`
2. `setsockopt(IP_MULTICAST_TTL, 1)`
3. `sendto()` verso indirizzo gruppo
4. `close()`

Il sender non si iscrive al gruppo — a meno che non voglia ricevere anche i propri pacchetti.

Opzioni `setsockopt` più importanti

Opzione	Significato
`IP_MULTICAST_TTL`	TTL dei pacchetti in uscita (def. 1)
`IP_MULTICAST_LOOP`	Il sender riceve i propri pacchetti? (def. 1 = sì)
`IP_MULTICAST_IF`	Scegli l'interfaccia di uscita esplicitamente
`IP_ADD_MEMBERSHIP`	Iscrizione al gruppo su un'interfaccia
`IP_DROP_MEMBERSHIP`	Uscita dal gruppo
`SO_REUSEADDR`	Più processi sulla stessa porta

Nota su IP_MULTICAST_LOOP: se è abilitato (default), il sender riceve i propri pacchetti multicast tramite il loopback del kernel — utile per test su una sola macchina.

Multicast vs Broadcast — riepilogo

	Broadcast	**Multicast**
Destinatari	Tutti sulla LAN	Solo iscritti al gruppo
IPv6	Non esiste	Sì (sostituisce broadcast)
Instradamento	Solo link-local	Possibile su WAN
Iscrizione richiesta	No	Sì (`IP_ADD_MEMBERSHIP`)
Filtraggio HW	No	Sì (IGMP snooping)
Protocollo segnalazione	—	IGMP

Cosa costruirete nell'HO

1. Compilare ed eseguire `sender.c` e `receiver.c` già funzionanti
2. Aprire più istanze del receiver e verificare che tutti ricevano
3. Disabilitare il loopback (`IP_MULTICAST_LOOP = 0`) e osservare
4. Costruire una **chat multicast**: ogni peer è sender e receiver
5. Aggiungere il nome utente al payload
6. (Opzionale) Implementare un sistema di *presenza*: chi è online?

Gruppo multicast: `239.0.0.1` **Porta:** `5000` **TTL:** `1`