

LABORATORIO DI RETI E SISTEMI DISTRIBUITI

HANDS-ON 6

Client multicast

Antonio Mastrolembro Ventura

543644

March 21, 2025

Indice

1	Introduzione	2
2	Definizione del problema	2
3	Metodologia	3
3.1	Socket UDP	3
3.2	Riutilizzo della porta	3
3.3	Configurazione dell'indirizzo e binding	4
3.4	Configurazione dell'indirizzo multicast	4
3.5	Aggiunta del client al gruppo multicast	5
3.6	Ricezione dei dati dai client	5
3.7	Uscita del client dal gruppo multicast	6
4	Presentazione dei risultati	6
4.1	Osservazioni	6
5	Conclusioni	7

1 Introduzione

Nelle reti dei calcolatori, la comunicazione **multicast** è una tecnica che permette a un mittente di inviare un messaggio/dato ad un gruppo di destinatari interessati contemporaneamente. A differenza di unicast, in cui i dati vengono inviati a un singolo destinatario, o broadcast, in cui i dati vengono inviati a tutti i dispositivi su una rete, il multicasting invia dati solo a un gruppo selezionato di ricevitori interessati alle informazioni. Questo approccio è particolarmente utile in contesti come lo streaming video, i giochi online, reti di sensori IoT, ecc.

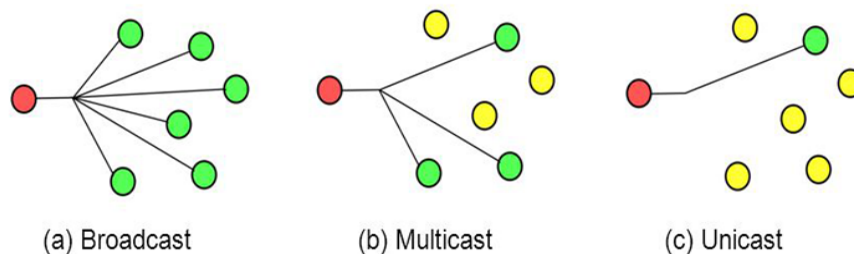


Figure 1: Broadcast vs Multicast vs Unicast

2 Definizione del problema

Il problema affrontato in questo hands-on riguarda la realizzazione di un **client multicast** il cui compito è quello di ricevere delle stringhe di testo inviate da un server. Per fare ciò, il client dovrà "registrarsi" ad un **gruppo multicast** e la rete si occuperà di consegnare i pacchetti multicast a tutti quelli che si sono registrati. Il client, dopo aver ricevuto cinque messaggi (indicativo) da parte del server, abbandonerà il gruppo multicast. Dovrà infine essere possibile lanciare più istanze del client che si aggiungano come membri al gruppo multicast.

3 Metodologia

Lo sviluppo del problema prevede la realizzazione di un programma in C che, attraverso una serie di passi ben definiti, garantisca il corretto funzionamento della comunicazione multicast.

3.1 Socket UDP

```
1 // Creo un socket UDP
2 if ((sockfd = socket(AF_INET, SOCK_DGRAM, 0)) < 0) {
3     perror("Socket creation failed!");
4     exit(EXIT_FAILURE);
5 }
```

Il primo passo consiste nella creazione di un socket UDP, poiché il multicast si basa su protocolli unidirezionali e senza controllo di connessione. Questo implica il fatto che i pacchetti vengono inviati dal mittente a più destinatari senza instaurare connessioni dirette.

3.2 Riutilizzo della porta

```
1 int reuse = 1;
2 if (setsockopt(sockfd, SOL_SOCKET, SO_REUSEADDR, &reuse, sizeof(
reuse)) < 0) {
3     perror("Setsockopt failed!");
4     close(sockfd);
5     exit(EXIT_FAILURE);
6 }
```

In questa parte di codice andiamo a fare una cosa molto importante: abilitiamo il riutilizzo della porta per il socket UDP.

Cosa accade normalmente? In genere quando un processo effettua un bind su una porta specifica, quella porta viene bloccata e nessun altro processo può "legarsi" ad essa. Nel nostro caso questa cosa non va bene in quanto più client devono ricevere lo stesso traffico multicast, dunque è necessario che più pro-

cessi possano effettuare il bind sulla stessa porta. L'opzione `SO_REUSEADDR` di `setsockopt` fa esattamente questo, permettendo a più socket di essere associati alla stessa combinazione di indirizzo e porta.

3.3 Configurazione dell'indirizzo e binding

```
1  memset(&local_addr, 0, sizeof(local_addr));
2  local_addr.sin_family = AF_INET;
3  local_addr.sin_port = htons(PORT);
4  local_addr.sin_addr.s_addr = htonl(INADDR_ANY);
5
6  if (bind(sockfd, (struct sockaddr *)&local_addr, sizeof(
local_addr)) < 0) {
7      perror("Bind failed!");
8      close(sockfd);
9      exit(EXIT_FAILURE);
10 }
```

Questa parte di codice si occupa della configurazione dell'indirizzo locale e collega il socket alla porta su cui riceverà i messaggi multicast. Il binding viene fatto su `INADDR_ANY` poichè permette al socket di accettare pacchetti multicast provenienti da qualsiasi interfaccia di rete disponibile sulla macchina.

3.4 Configurazione dell'indirizzo multicast

```
1  multicast_req.imr_multiaddr.s_addr = inet_addr(MULTICAST_GROUP);
2  multicast_req.imr_interface.s_addr = htonl(INADDR_ANY);
```

A questo punto è possibile configurare il socket per ricevere pacchetti dal gruppo multicast. La configurazione avviene tramite la struttura `ip_mreq` che possiamo trovare in:

```
$ cd /usr/include/netinet/in.h
```

Contiene due campi:

- `struct in_addr imr_multiaddr` in cui andiamo a mettere l'indirizzo del gruppo multicast a cui il processo vuole iscriversi;

- `struct in_addr imr_interface` in cui specifichiamo l'interfaccia di rete da utilizzare per la ricezione dei pacchetti multicast.

3.5 Aggiunta del client al gruppo multicast

```
1  if (setsockopt(sockfd, IPPROTO_IP, IP_ADD_MEMBERSHIP, &
    multicast_req, sizeof(multicast_req)) < 0) {
2      perror("Setsockopt failed!");
3      close(sockfd);
4      exit(EXIT_FAILURE);
5  }
```

Una volta configurato l'indirizzo multicast, possiamo aggiungere il client al gruppo. Attraverso la funzione `setsockopt` possiamo specificare l'opzione `IP_ADD_MEMBERSHIP` che ci permette di aggiungere il socket al gruppo multicast.

3.6 Ricezione dei dati dai client

```
1  while (msg_count < MAX_MESSAGES) {
2      int n = recvfrom(sockfd, buffer, sizeof(buffer) - 1, 0, NULL,
    NULL);
3      if (n < 0) {
4          perror("Recvfrom failed!");
5          break;
6      }
7      buffer[n] = '\0';
8      printf("Received [%d]: %s\n", msg_count + 1, buffer);
9      msg_count++;
10 }
```

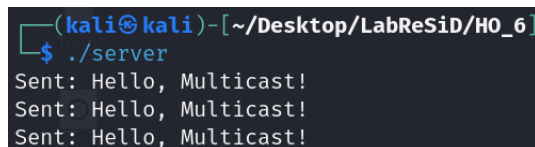
Adesso i client facenti parti del gruppo multicast, possono finalmente ricevere i dati che invia il server. Abbiamo un ciclo `while` che itera fino al raggiungimento di `MAX_MESSAGES` (impostato a 5) e che non fa altro che ricevere tramite `recvfrom()` il messaggio del server.

3.7 Uscita del client dal gruppo multicast

```
1  if (setsockopt(sockfd, IPPROTO_IP, IP_DROP_MEMBERSHIP, &
    multicast_req, sizeof(multicast_req)) < 0) {
2      perror("Setsockopt IP_DROP_MEMBERSHIP failed!");
3  } else {
4      printf("Abbandonato il gruppo multicast dopo %d messaggi.\n",
        MAX_MESSAGES);
5  }
```

Una volta che il client ha ricevuto per cinque volte il messaggio, richiamiamo la funzione `setsockopt` passando come opzione `IP_DROP_MEMBERSHIP` che, contrariamente a `IP_ADD_MEMBERSHIP`, rimuove il client dal gruppo multicast.

4 Presentazione dei risultati



```
(kali@kali)-[~/Desktop/LabReSiD/H0_6]
$ ./server
Sent: Hello, Multicast!
Sent: Hello, Multicast!
Sent: Hello, Multicast!
```

Figure 2: Server



```
(kali@kali)-[~/Desktop/LabReSiD/H0_6]
$ ./client
Received [1]: Hello, Multicast!
Received [2]: Hello, Multicast!
Received [3]: Hello, Multicast!
Received [4]: Hello, Multicast!
Received [5]: Hello, Multicast!
Abbandonato il gruppo multicast dopo 5 messaggi.
```

Figure 3: Client

Come possiamo vedere dall'output dei programmi, il server invia indefinitamente una stringa di testo ed il/i client appartenente/i al gruppo multicast ricevono soltanto per cinque volte il messaggio, dopodiché abbandonano.

4.1 Osservazioni

Durante il test è emersa una particolarità riguardante il binding sullo stesso indirizzo che mette a confronto due sistemi operativi differenti: macOS e Linux. Per quanto riguarda Linux, è possibile effettuare il binding sullo stesso indirizzo grazie al flag `SO_REUSEADDR`, in questo modo ho la possibilità di avere più client connessi e partecipanti al gruppo multicast. Su macOS invece il tutto

funziona in modo differente, poichè se tento di avviare più di un client ottengo un errore di questo tipo:

```
Bind failed!: Address already in use
```

Il motivo di questo errore è legato al fatto che il flag `SO_REUSEADDR` è gestito in modo differente fra i due sistemi operativi:

- Su Linux, il flag `SO_REUSEADDR` permette a più socket UDP di effettuare il binding sullo stesso indirizzo multicast e sulla stessa porta.
- Su macOS (e BSD in generale), il flag `SO_REUSEADDR` non ha lo stesso comportamento per gli indirizzi multicast. Infatti non permette a più socket di fare il binding sulla stessa coppia <indirizzo, porta> per UDP, a meno che non sia usato il flag `SO_REUSEPORT`, il quale consente esplicitamente questa condivisione.

Il comportamento di `SO_REUSEADDR` e `SO_REUSEPORT` nei differenti sistemi operativi è spiegato in dettaglio al seguente [link](#).

5 Conclusioni

In questo hands-on abbiamo visto e analizzato il funzionamento della comunicazione multicast, l'implementazione di un client multicast in C e il relativo test. Il client rispetta le caratteristiche di questo tipo di comunicazione: è capace di unirsi ad un gruppo multicast, ricevere messaggi da un server e successivamente abbandonare il gruppo. Questo tipo di esercizio ci dimostra come sia effettivamente possibile distribuire informazioni a più destinatari senza la necessità di instaurare delle connessioni point-to-point. Infine, durante il processo di testing, abbiamo visto come più sistemi operativi differenti gestiscono in modo diverso l'inizializzazione della comunicazione multicast, con la conseguente necessità di utilizzare approcci leggermente diversi in base alla macchina su cui vogliamo implementare un servizio di questo tipo.