Multicast (UDP)

Related pages: Unicast (UDP / TCP) Q&A

Why multicast

Multicast is useful when you have to transmit the SAME message to more than one host.

Usually the client that send multicast does not know how many servers will really receive his packets.
When talking about client-server in network, the **client** sends the request, the **server** receives the request and might send back an answer.

Since multicast is based UDP, the transmission is by default not reliable.

The advantage of using multicast instead of broadcast is that only interested hosts will get the message, and the message should be transmitted only once for many clients (saves a lot of bandwith).
Another advantage is the possibility of sending packets larger than interface MTU

Multicast on LAN

This is topic is quite complex, but we can for simplicity devide LANs in 3 kind:

1. LAN with hubs only
2. LAN with switches without IGMP support
3. LAN with switches with IGMP support

The first 2 types of LANs are the easiest to explain, since the behaviour for multicast is exactly the same; multicast is transmitted over all network segments.

In the third case where the LAN has IGMP aware switches, and IGMP support is enabled, multicast packets will be transmitted only on segments where hosts have requested it.
Other segments will not see this kind of traffice until one of the hosts requests it.
Of course packets sent to 224.0.0.1/224.0.0.2 could be observed on all segments, since these addresses have a special meaning (Reserved Multicast IP addresses)

Multicast on IPv6

IPv6 multicast (FF00::/8) addressing is defined in this document IPv6 address range, broadcast in IPv6 is a special multicast, address FF02:0:0:0:0:0:0:1 is IPv6 form of 255.255.255.255 in IPv4, it targets all hosts in local network (FF02::/16).

Multicast on WAN

... Still Working ....

What is multicast

Multicast is a kind of UDP traffic similar to BROADCAST, but only hosts that have explicitly requested to receive this kind of traffic will get it. This means that you have to JOIN a multicast group if you want to receive traffic that belongs to that group.

IP addresses in the range 224.0.0.0 to 239.255.255.255 (Class D addresses) belongs to multicast. No host can have this as IP address, but every machine can join a multicast address group.

Before you begin

- Multicast traffic is only UDP (not reliable)
- Multicast migth be 1 to many or 1 to none
- Not all networks are multicast enabled (Some routers do not forward Multicast)

Sample code

This is a sample multicast server without error handling
Comments on code

```
----------- cut here ------------------

// Multicast Server
// written for LINUX
// Version 0.0.2
//
// Change: IP_MULTICAST_LOOP : Enable / Disable loopback for outgoing messages
//
// Compile : gcc -o server server.c
//
// This code has NOT been tested
//

#include <stdio.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>

#define MAXBUFSIZE 65536 // Max UDP Packet size is 64 Kbyte

int main()
{
   int sock, status, socklen;
   char buffer[MAXBUFSIZE];
   struct sockaddr_in saddr;
   struct ip_mreq imreq;

   // set content of struct saddr and imreq to zero
   memset(&saddr, 0, sizeof(struct sockaddr_in));
   memset(&imreq, 0, sizeof(struct ip_mreq));

   // open a UDP socket
   sock = socket(PF_INET, SOCK_DGRAM, IPPROTO_IP);
   if ( sock < 0 )
     perror("Error creating socket"), exit(0);

   saddr.sin_family = PF_INET;
   saddr.sin_port = htons(4096); // listen on port 4096
   saddr.sin_addr.s_addr = htonl(INADDR_ANY); // bind socket to any interface
```

```
   status = bind(sock, (struct sockaddr *)&saddr, sizeof(struct sockaddr_in));

   if ( status < 0 )
     perror("Error binding socket to interface"), exit(0);

   imreq.imr_multiaddr.s_addr = inet_addr("226.0.0.1");
   imreq.imr_interface.s_addr = INADDR_ANY; // use DEFAULT interface

   // JOIN multicast group on default interface
   status = setsockopt(sock, IPPROTO_IP, IP_ADD_MEMBERSHIP,
              (const void *)&imreq, sizeof(struct ip_mreq));

   socklen = sizeof(struct sockaddr_in);

   // receive packet from socket
   status = recvfrom(sock, buffer, MAXBUFSIZE, 0,
                     (struct sockaddr *)&saddr, &socklen);

   // shutdown socket
   shutdown(sock, 2);
   // close socket
   close(sock);

   return 0;
}



---------- cut here ------------------
```

This is a sample multicast [client](#) without error handling

```
---------- cut here ------------------

// Multicast Client
// written for LINUX
// Version 0.0.2
//
// Change: IP_MULTICAST_LOOP : Enable / Disable loopback for outgoing messages
//
// Compile : gcc -o client client.c
//
// This code has NOT been tested
//

#include <stdio.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>

#define MAXBUFSIZE 65536 // Max UDP Packet size is 64 Kbyte

int main()
{
   int sock, status, socklen;
   char buffer[MAXBUFSIZE];
   struct sockaddr_in saddr;
   struct in_addr iaddr;
   unsigned char ttl = 3;
   unsigned char one = 1;

   // set content of struct saddr and imreq to zero
   memset(&saddr, 0, sizeof(struct sockaddr_in));
   memset(&iaddr, 0, sizeof(struct in_addr));

   // open a UDP socket
   sock = socket(PF_INET, SOCK_DGRAM, 0);
   if ( sock < 0 )
```

```
      perror("Error creating socket"), exit(0);

   saddr.sin_family = PF_INET;
   saddr.sin_port = htons(0); // Use the first free port
   saddr.sin_addr.s_addr = htonl(INADDR_ANY); // bind socket to any interface
   status = bind(sock, (struct sockaddr *)&saddr, sizeof(struct sockaddr_in));

   if ( status < 0 )
     perror("Error binding socket to interface"), exit(0);

   iaddr.s_addr = INADDR_ANY; // use DEFAULT interface

   // Set the outgoing interface to DEFAULT
   setsockopt(sock, IPPROTO_IP, IP_MULTICAST_IF, &iaddr,
              sizeof(struct in_addr));

   // Set multicast packet TTL to 3; default TTL is 1
   setsockopt(sock, IPPROTO_IP, IP_MULTICAST_TTL, &ttl,
              sizeof(unsigned char));

   // send multicast traffic to myself too
   status = setsockopt(sock, IPPROTO_IP, IP_MULTICAST_LOOP,
                       &one, sizeof(unsigned char));

   // set destination multicast address
   saddr.sin_family = PF_INET;
   saddr.sin_addr.s_addr = inet_addr("226.0.0.1");
   saddr.sin_port = htons(4096);

   // put some data in buffer
   strcpy(buffer, "Hello world\n");

   socklen = sizeof(struct sockaddr_in);
   // receive packet from socket
   status = sendto(sock, buffer, strlen(buffer), 0,
                     (struct sockaddr *)&saddr, socklen);

   // shutdown socket
   shutdown(sock, 2);
   // close socket
   close(sock);

   return 0;
}



---------- cut here -----------------
```

Be careful because ...

Possible differences might be noted between different flavour of Unix.

In some implementation you might need to call setsockopt before calling bind.

Note that we do not BIND the server to a specific interface, but we JOIN the multicast group (IP_MULTICAST_JOIN) on a specific interface

The same appens on the client side, we do not BIND to an interface but we set the transmitting interface with IP_MULTICAST_IF.

IP Classes

Class Name	Address Bits	From ... To	Pourpose
Class A	0	0.0.0.0 - 127.255.255.255	Public IP address
Class B	10	128.0.0.0 - 191.255.255.255	Public IP address
Class C	110	192.0.0.0 - 223.255.255.255	Public IP address
Class D	1110	224.0.0.0 - 239.255.255.255	Multicast IP Addresses
Class E	11110	240.0.0.0 - 255.255.255.255	Reserved

Programming Documentation

If you are interested in getting more details about socket programming have a look at

- *W. R. Stevens* "*UNIX Network Programming (Volume I)*"

Network Numbering Documentation

For more details about IP address space usage see IANA
Reserved Multicast IP addresses
IANA assigned Numbers

---

last modification 13 December 2003