How to UDP multicast to multiple device

By Chris Lee and Eric Liu

Table of Contents

1. Introduction

Multicasting is the act of sending and forwarding a single packet to multiple hosts. This can be very useful in situations where your application needs to send the same data to a lot of different users, but don't necessarily need to know any information about those users.

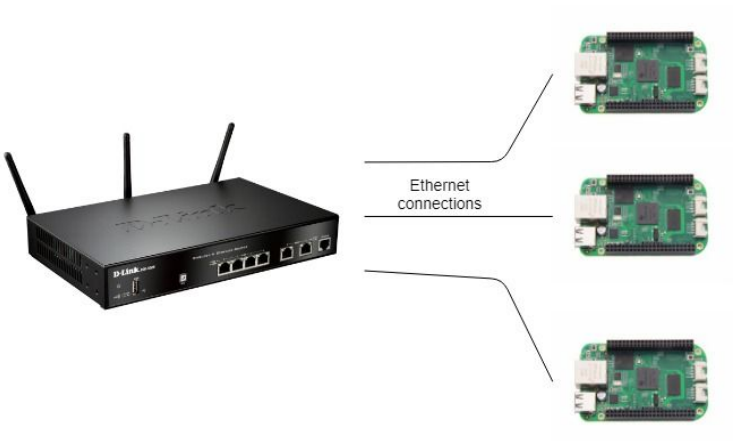
This guide explains how to multicast to a local network using C and socket programming.

1.1 Prerequisites

Multicasting requires that you have a router that supports IGMP (Internet Group Management Protocol), as this enables a packet to be forwarded to multiple interfaces connected to that router. Because there are a wide range of routers, we will not be demonstrating how to configure IGMP on your router.
In most cases, it is probably safe for you to assume that your router is able to handle IGMP packets.

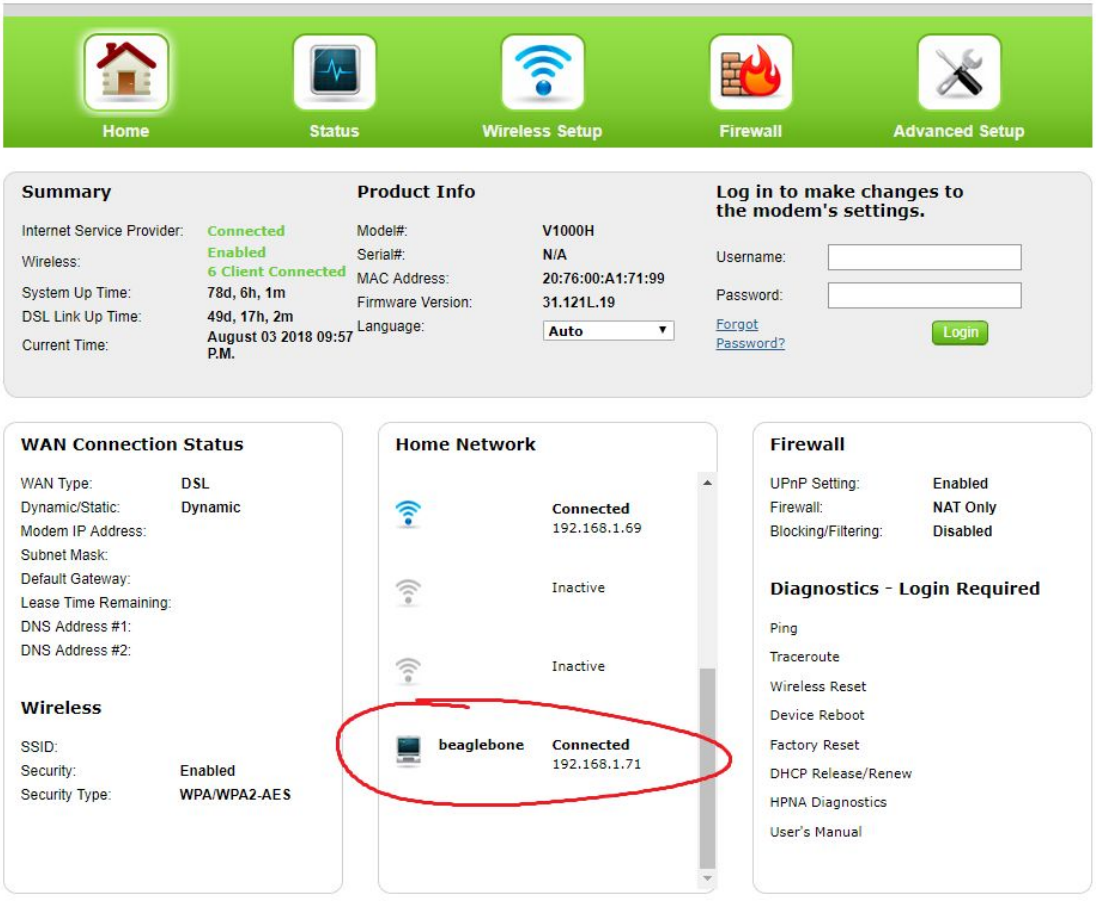
1.2 Physical setup

There are two options for setting this up. The first option is to connect each Beaglebone to the same router using an ethernet cable. This is a simple and straightforward approach, but is limited by the number of ethernet slots on the router and the length of the ethernet cable.



A second option is to use a USB WiFi adapter and connect to the same WiFi network. Note that if the WiFi connection requires a password, it first needs to be configured on the Beaglebone (out of the scope of this guide). It is also possible to mix and match wireless and wired connections as long as all devices are connected to the same network.

To ensure that the devices are on the same network, navigate to your router's settings page. The Beaglebones should be listed in the router's connections.



Finally, a minimal but functional approach would be to connect two Beaglebone devices directly with an ethernet cable. This forms a local network with two devices and they can send packets or multicast to each other.

2. Link Setup for Linux

On the device that will send the multicast packets, you may need to configure how that device will route the packets that are being sent to the multicast address.

For the Beaglebone Green, after getting the networking set up, packets are usually forwarded through the Ethernet-Over-USB connection. Instead, you want to make sure that you are sending packets via the Ethernet connection.

1. Before starting, choose an IP address to use for multicasting. Typically, this is an address between the range of 224.0.0.0 to 239.255.255.255.
    - Do not choose 224.0.0.1, 224.0.0.2, or 224.0.0.22. The first two addresses are used by IGMP to send packets to all hosts that support multicasting. In some cases, your router may prevent packets on these addresses from being sent. 224.0.0.22 is used by IGMP to manage multicast group memberships.
2. Find out if the interface which represents the connection between your device and the router will be handling the UDP packets being sent by your server. You can do this by running the `route` command in your terminal.

```
$ route
Kernel IP routing table
Destination     Gateway         Genmask         Flags Metric Ref    Use Iface
default         idunno          0.0.0.0         UG    100    0        0 enp0s3
link-local      *               255.255.0.0     U     1000   0        0 enx04a316eefc72
192.168.1.0     *               255.255.255.0   U     100    0        0 enp0s3
192.168.6.0     *               255.255.255.252 U     100    0        0 enx04a316eefc75
192.168.7.0     *               255.255.255.252 U     0      0        0 enx04a316eefc72
```

    - If using the IP address 224.0.0.10 in the situation above, you can see that the packet will be handled correctly. UDP multicast packets will be sent to the "`enp0s3`"interface because none of the other "Genmasks" will match with the 224.0.0.10 address.
    - On the Beaglebone Green with the default kernel, the interface you should be looking for would typically be named "eth0" when you are using Ethernet, or "wlan0" if you are using a WiFi dongle.
3. If your multicast packets are being sent to the incorrect interface, run the following command to update the routing table. Make sure that you are running the command with superuser permissions.

```
$ ip route add 224.0.0.0/4 dev eth0
```

    - You can replace "`eth0`" with the name of whichever interface you are using.

3. Server Programming in C

If you are familiar with UDP socket programming in C, the server should be simple to implement because the process is exactly the same any other server in C.

1. Import the following header files into your networking module:

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

```c
#include <netinet/in.h>
#include <arpa/inet.h>
```

2. Create a socket file descriptor for sending UDP datagrams. Optionally, you can bind this file descriptor to a port, but this isn't necessary.

```c
int sd = socket(AF_INET, SOCK_DGRAM, 0);
```

3. The file descriptor should now be ready to send UDP packets. This can be done using the "`sendto`" function.

```c
struct sockaddr_in addr;
ssize_t res;
unsigned int addrlen;

memset(&addr, 0, sizeof(addr));
addr.sin_family = AF_INET;
addr.sin_port = htons(MULTICAST_PORT);
addr.sin_addr.s_addr = inet_addr(MULTICAST_ADDR);
addrlen = sizeof(addr);

// (create message)

res = sendto(sd, message, message_size, 0,
             (struct sockaddr *) &addr, addrlen);
```

- Replace the "`MULTICAST_PORT`" macro with any available port number. This should be an int. Make sure it matches with the port number that you are going to bind to in your client program.
- Replace the "`MULTICAST_ADDR`" macro with the IP address you chose in step 1 of the previous section. This should be a string containing your IPv4 address.
- Replace the "`message`" variable with a buffer (`char *`) containing your message and "`message_size`" with the size of your buffer in bytes (`unsigned int`).
- The "`res`" variable isn't entirely necessary, but it is recommended to do error checking on it to ensure that your message is being sent correctly.

4. Client Programming in C

1. Follow steps 1 and 2 of the previous section to obtain a valid socket file descriptor.
2. Bind the file descriptor to your multicast port.

```c
struct sockaddr_in addr;
int res;

memset(&addr, 0, sizeof(addr));
addr.sin_family = AF_INET;
addr.sin_port = htons(MULTICAST_PORT);
addr.sin_addr.s_addr = htonl(INADDR_ANY);
```

```
res = bind(sd, (struct sockaddr *) &addr, sizeof(addr));
```

3. Use "`setsockopt`" to designate the multicast address your application wants to listen to. You must also specify which interface you want the multicasts to come from.

```
struct ip_mreq mreq;
int res;

mreq.imr_multiaddr.s_addr = inet_addr(MULTICAST_ADDR);
mreq.imr_interface.s_addr = inet_addr(INTERFACE_ADDR);

res = setsockopt(sd, IPPROTO_IP, IP_ADD_MEMBERSHIP, &mreq, sizeof(mreq));
```

- As with before, replace "`MULTICAST_ADDR`" with a string representing your chosen IPv4 multicast address.
- Replace "`INTERFACE_ADDR`" with the local address of the interface that is connected to the router. To find this, run the `` `ifconfig` `` command on your multicast receiver device and copy the value of "`inet addr`".

```
# ifconfig
enp0s3    Link encap:Ethernet  HWaddr 08:00:27:87:79:0d
          inet addr:192.168.1.103  Bcast:192.168.1.255  Mask:255.255.255.0
          inet6 addr: fe80::dac6:da2:9c80:56c0/64 Scope:Link
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
          RX packets:12874 errors:0 dropped:0 overruns:0 frame:0
          TX packets:3575 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:3352789 (3.3 MB)  TX bytes:422526 (422.5 KB)
```

4. You should now be able to receive multicasts using "`recvfrom`".

```
ssize_t res;
unsigned int addrlen;

addrlen = sizeof(addr);
res = recvfrom(sd, message, sizeof(message), 0,
               (struct sockaddr *) &addr, &addrlen);
```

- "`message`" should be a pointer to a character buffer that you are going to use to store the contents of the received multicast. The size of the buffer, represented by "`sizeof(message)`", should typically be above 1500 bytes, as most networks use that as the maximum packet size.
- You should do error checking with the "`res`" variable.

5. Troubleshooting

- On wireshark, if you are unable to see an IGMP packet being sent to your router by your client application, it is likely that your "`mreq.imr_interface.s_addr`" variable is assigned incorrectly. Make sure that it is not being set as "`htonl(INADDR_ANY)`".