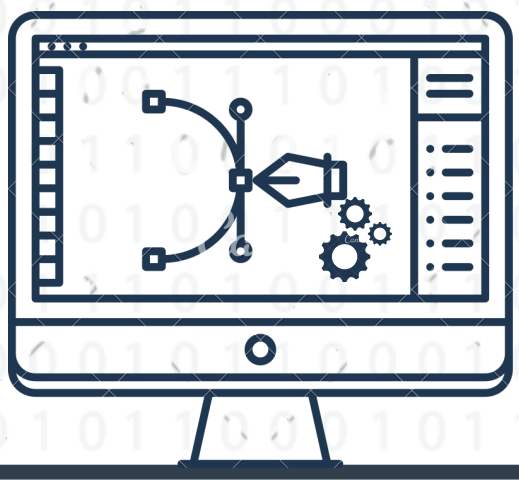
ESCUELA SUPERIOR DE COMPUTO



APLICACIONES PARA COMUNICACIONES EN RED

Tarea #4
Broadcast y multicast

Alumnos:
Caxantheje Ortiz Jazmin Lizeth
Lorenzo Pioquinto Alejandro
Rubio Haro Rodrigo R.

Profesor: Rangel Gonzalez Josue

CDMX. OCTUBRE, 2022.

INSTITUTO POLITÉCNICO NACIONAL

Multicast

El tráfico IP Multicast, o también conocido como multidifusión IP, es un método para transmitir información a un grupo de receptores (clientes) que están configurados para tal fin. Los equipos que no están configurados específicamente, no recibirán este tráfico de red y podrán dedicarse a enviar y recibir otro tipo de tráfico. En redes IPv4 existen un total de cuatro tipo de comunicaciones diferentes que se pueden realizar, estas son las siguientes:

Unicast

Es el tipo de comunicación más común, la dirección es unidifusión, es decir, desde un origen en concreto hasta un destino. Tenemos un único emisor y receptor, y se puede utilizar tanto para enviar como para recibir datos. Este tipo de comunicación es ampliamente usada, por ejemplo, para la navegación web, transferencia de archivos vía Samba o FTP, o casi cualquier otro tipo de comunicación. Si queremos enviar la misma información a varios usuarios, tendremos que enviar los datos una vez a cada uno de los receptores.

Broadcast

Este tipo de comunicación permite enviar los datos a todos los usuarios que hay en la misma red local. Podremos enviar un mensaje a la dirección IP de broadcast (que es la última dirección IP de una subred) y automáticamente el resto de usuarios conectados recibirán esta comunicación. Tenemos una dirección IP especial que es la 255.255.255.255 que representa un broadcast a toda la red local, esta dirección IP es ampliamente usada cuando enviamos un mensaje DHCP Discovery, para intentar descubrir dónde está ubicado el servidor DHCP en la red.

Anycast

Este tipo de comunicación es de uno a muchos, sin embargo, los datos no son transmitidos a todos los receptores, solamente lo enviarán a los más cercanos. Este método es el usado por los servidores DNS para balancear el tráfico de datos entre los diferentes servidores que hay repartidos por todo el mundo. Gracias a las IP Anycast, un mismo servidor DNS (8.8.8.8, por ejemplo) puede tener esta dirección tanto en España como en EEUU, ya que los protocolos de enrutamiento dinámicos se encargarán de enviar la petición al servidor DNS más cercano.

Ejemplo de Implementación de Chat Multicast

```
// FILE: SERVER
#ifdef _WIN32
    #include <Winsock2.h> // before Windows.h, else Winsock 1 conflict
    #include <Ws2tcpip.h> // needed for ip_mreq definition for multicast
    #include <Windows.h>
#else
    #include <sys/types.h>
    #include <sys/socket.h>
    #include <netinet/in.h>
    #include <arpa/inet.h>
    #include <unistd.h> // for sleep()
#endif

#include <string.h>
#include <stdio.h>
#include <stdlib.h>

int main(int argc, char *argv[])
{
    if (argc != 3) {
       printf("Command line args should be multicast group and port\n");
       printf("(e.g. for SSDP, `sender 239.255.255.250 1900`)\n");
       return 1;
    }

    char* group = argv[1]; // e.g. 239.255.255.250 for SSDP
    int port = atoi(argv[2]); // 0 if error, which is an invalid port

    // !!! If test requires, make these configurable via args
    //
    const int delay_secs = 1;
    const char *message = "Hello, World!";

#ifdef _WIN32
    //
    // Initialize Windows Socket API with given VERSION.
    //
    WSADATA wsaData;
    if (WSAStartup(0x0101, &wsaData)) {
        perror("WSAStartup");
        return 1;
    }
#endif

    // create what looks like an ordinary UDP socket
    //
    int fd = socket(AF_INET, SOCK_DGRAM, 0);
    if (fd < 0) {
        perror("socket");
        return 1;
    }
```

```
    // set up destination address
    struct sockaddr_in addr;
    memset(&addr, 0, sizeof(addr));
    addr.sin_family = AF_INET;
    addr.sin_addr.s_addr = inet_addr(group);
    addr.sin_port = htons(port);

      // now just sendto() our destination!
    //
    while (1) {
        char ch = 0;
        int nbytes = sendto(
            fd,
            message,
            strlen(message),
            0,
            (struct sockaddr*) &addr,
            sizeof(addr)
        );
        if (nbytes < 0) {
            perror("sendto");
            return 1;
        }

     #ifdef _WIN32
          Sleep(delay_secs * 1000); // Windows Sleep is milliseconds
     #else
          sleep(delay_secs); // Unix sleep is seconds
     #endif
     }

#ifdef _WIN32
    WSACleanup();
#endif

    return 0;
}
```

```
// FILE: CLIENT
#ifdef _WIN32
    #include <Winsock2.h> // before Windows.h, else Winsock 1 conflict
    #include <Ws2tcpip.h> // needed for ip_mreq definition for multicast
    #include <Windows.h>
#else
    #include <sys/types.h>
    #include <sys/socket.h>
    #include <netinet/in.h>
    #include <arpa/inet.h>
    #include <time.h>
#endif

#include <string.h>
#include <stdio.h>
#include <stdlib.h>

#define MSGBUFSIZE 256

int main(int argc, char *argv[])
{
    if (argc != 3) {
       printf("Command line args should be multicast group and port\n");
       printf("(e.g. for SSDP, `listener 239.255.255.250 1900`)\n");
       return 1;
    }

    char* group = argv[1]; // e.g. 239.255.255.250 for SSDP
    int port = atoi(argv[2]); // 0 if error, which is an invalid port


#ifdef _WIN32
    // Initialize Windows Socket API with given VERSION.
    //
    WSADATA wsaData;
    if (WSAStartup(0x0101, &wsaData)) {
        perror("WSAStartup");
        return 1;
    }
#endif

    // create what looks like an ordinary UDP socket
    int fd = socket(AF_INET, SOCK_DGRAM, 0);
    if (fd < 0) {
        perror("socket");
        return 1;
    }
    u_int yes = 1;
    if (setsockopt(fd, SOL_SOCKET, SO_REUSEADDR, (char*) &yes, sizeof(yes)) < 0 ){
       perror("Reusing ADDR failed");
       return 1;
    }

        // set up destination address
```

```
//
    struct sockaddr_in addr;
    memset(&addr, 0, sizeof(addr));
    addr.sin_family = AF_INET;
    addr.sin_addr.s_addr = htonl(INADDR_ANY); // differs from sender
    addr.sin_port = htons(port);

    // bind to receive address
    //
    if (bind(fd, (struct sockaddr*) &addr, sizeof(addr)) < 0) {
        perror("bind");
        return 1;
    }

    // use setsockopt() to request that the kernel join a multicast group
    //
    struct ip_mreq mreq;
    mreq.imr_multiaddr.s_addr = inet_addr(group);
    mreq.imr_interface.s_addr = htonl(INADDR_ANY);
    if (
        setsockopt(
            fd, IPPROTO_IP, IP_ADD_MEMBERSHIP, (char*) &mreq, sizeof(mreq)
        ) < 0
    ){
        perror("setsockopt");
        return 1;
    }

    // now just enter a read-print loop
    //
    while (1) {
        char msgbuf[MSGBUFSIZE];
        int addrlen = sizeof(addr);
        int nbytes = recvfrom(
            fd,
            msgbuf,
            MSGBUFSIZE,
            0,
            (struct sockaddr *) &addr,
            &addrlen
        );
        if (nbytes < 0) {
            perror("recvfrom");
            return 1;
        }
        msgbuf[nbytes] = '\0';
        puts(msgbuf);
     }

#ifdef _WIN32
    WSACleanup();
#endif

    return 0;
}
```