



Instituto Politécnico Nacional

IPN

Escuela Superior de Cómputo

ESCOM

Academia de Redes Aplicaciones para comunicaciones  
en red

6CV2

Práctica 9

“UDP Multicast”

Alumna:

Navarrete Becerril Sharon Anette

Fecha de entrega: “ 25-Noviembre-2024”

Profesor: Ojeda Santillan Rodrigo

## OBJETIVO

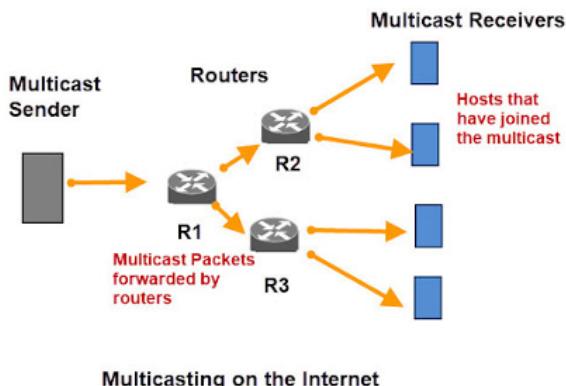
Se busca en la presente práctica enviar un mensaje al servidor 1 y al servidor 2 que se encuentran en ejecución en un grupo multicast, desde el cliente al mandar un mensaje su funcionamiento sería recibir ese mensaje en el grupo y puerto asociado a ese servidor. Se realizará en la dirección multicast siguiente: 224.1.1.1 En el puerto: 5004

## INTRODUCCIÓN

Multicast es una técnica de comunicación en redes donde un solo flujo de datos es enviado de manera eficiente a múltiples receptores interesados. A diferencia de unicast, que requiere enviar copias individuales a cada receptor, multicast permite que los datos sean transmitidos simultáneamente a un grupo de receptores utilizando una dirección IP especial.

Las direcciones IP de multicast se encuentran dentro de rangos específicos tanto en IPv4 como en IPv6:

- **IPv4:** De 224.0.0.0 a 239.255.255.255.
- **IPv6:** Comienzan con ff00::/8.



Algunas de sus características son:

- **Uso eficiente del ancho de banda:** Permite enviar un solo flujo de datos que los routers replican solo cuando es necesario, ahorrando ancho de banda en comparación con unicast.
- **Escalabilidad:** Es altamente escalable, permitiendo que muchos receptores reciban datos al mismo tiempo sin sobrecargar al emisor.
- **Direcciones IP especiales:** Utiliza rangos de direcciones IP reservadas para multicast. Los dispositivos interesados deben unirse a estos grupos configurando sus interfaces de red.
- **Protocolos de gestión:** Protocolos como IGMP (para IPv4) y MLD (para IPv6) gestionan la membresía en grupos multicast.
- **Enrutamiento multicast:** Los routers deben soportar protocolos de enrutamiento como PIM para distribuir los flujos de datos multicast eficientemente.

## DESARROLLO

Primero, se definirá la lógica a utilizar para el cliente, donde se emplea la biblioteca socket para enviar mensajes a un grupo de multidifusión. Se configura la dirección de multidifusión en 224.1.1.1 y el puerto 5004, estableciendo un TTL (Time to Live) de 2 saltos en la red. A continuación, se crea un socket UDP (usando socket.AF\_INET para IPv4 y socket.SOCK\_DGRAM para UDP) y se ajusta la opción del socket para definir el TTL mediante setsockopt. El código entra en un bucle que solicita al usuario que ingrese mensajes, los cuales se convierten a bytes (encode('utf-8')) y se envían al grupo de multidifusión con sendto. Si el usuario escribe 'salir', el bucle se interrumpe y el programa finaliza, permitiendo al usuario enviar múltiples mensajes hasta que desee detener el programa. Una vez entendido el funcionamiento del cliente, se debe agregar el siguiente fragmento de código en el archivo cliente.py.

### Código del cliente:

```
import socket

# Configuración del grupo de multidifusión y el puerto
group = '224.1.1.1'
port = 5004

# Restricción de 2 saltos en la red
ttl = 2

# Crear el socket UDP
sock = socket.socket(socket.AF_INET, socket.SOCK_DGRAM, socket.IPPROTO_UDP)
sock.setsockopt(socket.IPPROTO_IP, socket.IP_MULTICAST_TTL, ttl)

print("Escribe tus mensajes. Escribe 'salir' para terminar.")
try:
    while True:
        # Pedir al usuario que ingrese un mensaje
        message = input("Ingrese el mensaje a enviar: ")
        # Salir del bucle si el usuario escribe 'salir'
        if message.lower() == 'salir':
            break
        # Enviar el mensaje al grupo de multidifusión
        sock.sendto(message.encode('utf-8'), (group, port))
except Exception as e:
    print(f"Error: {e}")
finally:
    sock.close()
    print("Cliente cerrado.")
```

Para establecer la lógica del servidor, primero es importante entender su funcionalidad. Para configurar el servidor de manera que reciba mensajes de un grupo de multidifusión, se deben importar las bibliotecas socket y struct. Se define la dirección de multidifusión como '224.1.1.1' y el puerto como 5004. A continuación, se crea un socket UDP utilizando socket.AF\_INET para IPv4 y socket.SOCK\_DGRAM para UDP. El socket se configura para permitir la reutilización de direcciones mediante setsockopt y se vincula al puerto de multidifusión especificado. Con struct.pack, se prepara una solicitud de membresía al grupo de multidifusión y se agrega al socket mediante setsockopt. Luego, se imprime un mensaje indicando que el servidor está listo para funcionar. En un bucle infinito, el servidor recibe mensajes de hasta 10240 bytes y los muestra en la consola, permitiendo recibir y visualizar todos los mensajes enviados a la dirección y puerto de multidifusión configurados.

Posteriormente, se debe añadir el siguiente fragmento de código en el archivo servidor\_1.py para completar su funcionamiento.

### Código del servidor:

```
import socket
import struct

# Configuración del grupo de multidifusión y el puerto
MCAST_GRP = '224.1.1.1'
MCAST_PORT = 5004

# Crear socket UDP
sock = socket.socket(socket.AF_INET, socket.SOCK_DGRAM, socket.IPPROTO_UDP)
sock.setsockopt(socket.SOL_SOCKET, socket.SO_REUSEADDR, 1)
sock.bind(('', MCAST_PORT))

# Unirse al grupo de multidifusión
mreq = struct.pack("4sl", socket.inet_aton(MCAST_GRP), socket.INADDR_ANY)
sock.setsockopt(socket.IPPROTO_IP, socket.IP_ADD_MEMBERSHIP, mreq)

print("Servidor iniciado...")
try:
    while True:
        data, addr = sock.recvfrom(1024) # Tamaño del buffer ajustado
        print(f"Mensaje recibido desde {addr}: {data.decode('utf-8')}")
except KeyboardInterrupt:
    print("\nServidor detenido.")
finally:
    sock.close()
```

Finalmente, se debe añadir el siguiente fragmento de código en el archivo servidor\_2.py para completar su funcionamiento.

### Código del servidor:

```
import socket
import struct

# Configuración del grupo de multidifusión y el puerto
MCAST_GRP = '224.1.1.1'
MCAST_PORT = 5004

# Crear socket UDP
sock = socket.socket(socket.AF_INET, socket.SOCK_DGRAM, socket.IPPROTO_UDP)
sock.setsockopt(socket.SOL_SOCKET, socket.SO_REUSEADDR, 1)
sock.bind(('', MCAST_PORT))

# Unirse al grupo de multidifusión
mreq = struct.pack("4sl", socket.inet_aton(MCAST_GRP), socket.INADDR_ANY)
sock.setsockopt(socket.IPPROTO_IP, socket.IP_ADD_MEMBERSHIP, mreq)

print("Servidor iniciado...")
```

```
try:  
    while True:  
        data, addr = sock.recvfrom(1024) # Tamaño del buffer ajustado  
        print(f"Mensaje recibido desde {addr}: {data.decode('utf-8')}")  
except KeyboardInterrupt:  
    print("\nServidor detenido.")  
finally:  
    sock.close()
```

## Capturas de funcionamiento

The image consists of three vertically stacked screenshots of a Windows Command Prompt window. Each screenshot shows the output of a different Python script.

- Screenshot 1:** Shows the execution of `python servidor_1.py`. The output indicates the server has started and is receiving messages from an IP address. The messages received are "holaaaaaaaa", "como estas", and "bye".
- Screenshot 2:** Shows the execution of `python servidor_2.py`. The output indicates the server has started and is receiving messages from an IP address. The messages received are "holaaaaaaaa", "como estas", and "bye".
- Screenshot 3:** Shows the execution of `python cliente.py`. The program prompts the user to enter messages. The user enters "holaaaaaaaa", "como estas", "bye", and "salir". The program then closes with the message "Cliente cerrado."

## Análisis de Wireshark

\*Adapter for loopback traffic capture

Archivo Edición Visualización Ir Captura Analizar Estadísticas Telefonía Wireless Herramientas Ayuda

Aplique un filtro de visualización ... <Ctrl-/>

No.	Time	Source	Destination	Protocol	Lengt!	Info
722	13.125838	127.0.0.1	127.0.0.1	TCP	44	56449 + 56450 [ACK] Seq=1 Ack=168 Win=8321 Len=0
723	13.129011	127.0.0.1	127.0.0.1	TCP	45	49731 + 49732 [PSH, ACK] Seq=167 Ack=1 Win=8442 Len=1
724	13.129030	127.0.0.1	127.0.0.1	TCP	44	49731 + 49731 [ACK] Seq=1 Ack=168 Win=8258 Len=0
725	13.129148	127.0.0.1	127.0.0.1	TCP	45	56450 + 56449 [PSH, ACK] Seq=168 Ack=1 Win=8442 Len=1
726	13.129169	127.0.0.1	127.0.0.1	TCP	44	56449 + 56450 [ACK] Seq=1 Ack=169 Win=8321 Len=0
727	13.160258	127.0.0.1	127.0.0.1	TCP	45	49731 + 49732 [PSH, ACK] Seq=168 Ack=1 Win=8442 Len=1
728	13.160303	127.0.0.1	127.0.0.1	TCP	44	49731 + 49731 [ACK] Seq=1 Ack=169 Win=8258 Len=0
729	13.818729	172.100.92.160	224.1.1.1	UDP	43	60338 + 5004 Len=11
730	14.852357	127.0.0.1	127.0.0.1	TCP	56	48684 + 49746 [SYN] Seq=0 Win=65535 MSS=65495 WS=256 SACK_PERM
731	14.852429	127.0.0.1	127.0.0.1	TCP	56	49746 + 48684 [SYN, ACK] Seq=0 Ack=1 Win=65535 Len=MSS=65495 WS=256 SACK_PERM
732	14.852468	127.0.0.1	127.0.0.1	TCP	44	48684 + 49746 [ACK] Seq=1 Ack=1 Win=2161152 Len=0
733	14.852744	127.0.0.1	127.0.0.1	TCP	100	48684 + 49746 [PSH, ACK] Seq=1 Ack=1 Win=2161152 Len=56
734	14.852766	127.0.0.1	127.0.0.1	TCP	44	49746 + 48684 [ACK] Seq=1 Ack=57 Win=2161152 Len=0
735	14.852808	127.0.0.1	127.0.0.1	TCP	221	48684 + 49746 [PSH, ACK] Seq=57 Ack=1 Win=2161152 Len=177
736	14.852823	127.0.0.1	127.0.0.1	TCP	44	49746 + 48684 [ACK] Seq=1 Ack=234 Win=2160896 Len=0
737	15.850260	127.0.0.1	127.0.0.1	TCP	100	49747 + 49775 [PSH, ACK] Seq=57 Ack=57 Win=8436 Len=56
738	15.850319	127.0.0.1	127.0.0.1	TCP	44	49775 + 49747 [ACK] Seq=57 Ack=13 Win=8427 Len=0
739	15.850325	127.0.0.1	127.0.0.1	TCP	100	49747 + 49749 [PSH, ACK] Seq=57 Ack=57 Win=8434 Len=56
739	15.850325	127.0.0.1	127.0.0.1	TCP	100	49747 + 49749 [PSH, ACK] Seq=57 Ack=57 Win=8434 Len=56
739	15.850325	127.0.0.1	127.0.0.1	TCP	100	49747 + 49749 [PSH, ACK] Seq=57 Ack=57 Win=8434 Len=56
> Frame 729: 43 bytes on wire (344 bits), 43 bytes captured (344 bits) on interface \Device\NPF_Loopback, 0x0000 02 00 00 45 00 00 27 15 91 00 00 02 11 00 00 ... E... > Null/Loopback > Internet Protocol Version 4, Src: 172.100.92.160, Dst: 224.1.1.1 > User Datagram Protocol, Src Port: 60338, Dst Port: 5004 > Data (11 bytes)	0010	ac 64 5c a0 e0 01 01 eb b2 13 8c 00 13 bc 8c d\..... 0020	68 6f 6c 61 61 61 61 61 holaaaaaa aaa			

\*Adapter for loopback traffic capture

Archivo Edición Visualización Ir Captura Analizar Estadísticas Telefonía Wireless Herramientas Ayuda

Aplique un filtro de visualización ... <Ctrl-/>

No.	Time	Source	Destination	Protocol	Lengt!	Info
736	14.852823	127.0.0.1	127.0.0.1	TCP	44	49746 + 64864 [ACK] Seq=1 Ack=234 Win=2160896 Len=0
737	15.850260	127.0.0.1	127.0.0.1	TCP	100	49747 + 49775 [PSH, ACK] Seq=57 Ack=57 Win=8436 Len=56
738	15.850319	127.0.0.1	127.0.0.1	TCP	44	49775 + 49747 [ACK] Seq=57 Ack=113 Win=8427 Len=0
739	15.850325	127.0.0.1	127.0.0.1	TCP	100	49747 + 49749 [PSH, ACK] Seq=57 Ack=8434 Len=56
740	15.850370	127.0.0.1	127.0.0.1	TCP	44	49749 + 49747 [ACK] Seq=57 Ack=113 Win=1261 Len=0
741	15.865832	127.0.0.1	127.0.0.1	TCP	100	49775 + 49747 [PSH, ACK] Seq=57 Ack=113 Win=8427 Len=56
742	15.865875	127.0.0.1	127.0.0.1	TCP	44	49747 + 49775 [ACK] Seq=113 Ack=113 Win=8435 Len=0
743	15.865966	127.0.0.1	127.0.0.1	TCP	100	49749 + 49747 [PSH, ACK] Seq=57 Ack=113 Win=1261 Len=56
744	15.866011	127.0.0.1	127.0.0.1	TCP	44	49747 + 49749 [ACK] Seq=113 Ack=113 Win=8433 Len=0
745	16.073502	172.100.92.160	172.100.92.160	ICMP	84	Destination unreachable (Host unreachable)
746	16.530084	172.100.92.160	224.1.1.1	UDP	42	60338 + 5004 Len=10
747	18.207817	127.0.0.1	127.0.0.1	TCP	100	48684 + 49746 [PSH, ACK] Seq=234 Ack=1 Win=2161152 Len=56
748	18.207850	127.0.0.1	127.0.0.1	TCP	44	49746 + 64864 [ACK] Seq=1 Ack=290 Win=2160896 Len=0
749	18.207871	127.0.0.1	127.0.0.1	TCP	222	48684 + 49746 [PSH, ACK] Seq=290 Ack=1 Win=2161152 Len=178
750	18.207880	127.0.0.1	127.0.0.1	TCP	44	49746 + 64864 [ACK] Seq=1 Ack=468 Win=2160640 Len=0
751	18.211562	127.0.0.1	127.0.0.1	TCP	100	49746 + 64864 [PSH, ACK] Seq=1 Ack=468 Win=2160640 Len=56
752	18.211602	127.0.0.1	127.0.0.1	TCP	44	48684 + 49746 [ACK] Seq=468 Ack=57 Win=2161152 Len=0
753	18.211638	127.0.0.1	127.0.0.1	TCP	126	49746 + 64864 [PSH, ACK] Seq=57 Ack=468 Win=2160640 Len=82
> Frame 746: 42 bytes on wire (336 bits), 42 bytes captured (336 bits) on interface \Device\NPF_Loopback, 0x0000 02 00 00 45 00 00 26 15 92 00 00 02 11 00 00 ... E...&... > Null/Loopback > Internet Protocol Version 4, Src: 172.100.92.160, Dst: 224.1.1.1 > User Datagram Protocol, Src Port: 60338, Dst Port: 5004 > Data (10 bytes)	0010	ac 64 5c a0 e0 01 01 eb b2 13 8c 00 12 50 58 d\..... 0020	63 6f 6d 6f 20 65 73 74 61 73 como est as			

\*Adapter for loopback traffic capture

Archivo Edición Visualización Ir Captura Analizar Estadísticas Telefonía Wireless Herramientas Ayuda

Aplique un filtro de visualización ... <Ctrl-/>

No.	Time	Source	Destination	Protocol	Lengt!	Info
785	19.214872	127.0.0.1	127.0.0.1	TCP	44	64866 + 64865 [ACK] Seq=10 Ack=2 Win=2161152 Len=0
786	19.215214	127.0.0.1	127.0.0.1	TCP	44	64865 + 64866 [FIN, ACK] Seq=2 Ack=10 Win=2161152 Len=0
787	19.215246	127.0.0.1	127.0.0.1	TCP	44	64866 + 64865 [ACK] Seq=10 Ack=3 Win=2161152 Len=0
788	19.215282	127.0.0.1	127.0.0.1	TCP	44	64866 + 64865 [RST, ACK] Seq=10 Ack=3 Win=0
789	19.947418	172.100.92.160	224.1.1.1	UDP	35	60338 + 5004 Len=3
790	23.465192	127.0.0.1	127.0.0.1	TCP	44	48684 + 49746 [FIN, ACK] Seq=1268 Ack=2759 Win=2158336 Len=0
791	23.465229	127.0.0.1	127.0.0.1	TCP	44	49746 + 64864 [ACK] Seq=2759 Ack=1269 Win=2159872 Len=0
792	23.481068	127.0.0.1	127.0.0.1	TCP	44	49746 + 64864 [FIN, ACK] Seq=2759 Ack=1269 Win=2159872 Len=0
793	23.481111	127.0.0.1	127.0.0.1	TCP	44	64864 + 49746 [ACK] Seq=1269 Ack=2760 Win=2158336 Len=0
794	25.188373	127.0.0.1	127.0.0.1	TCP	45	49731 + 49732 [PSH, ACK] Seq=169 Ack=1 Win=8442 Len=1
795	25.188415	127.0.0.1	127.0.0.1	TCP	44	49731 + 49731 [ACK] Seq=169 Ack=10 Win=8258 Len=0
796	25.188903	127.0.0.1	127.0.0.1	TCP	45	56450 + 56449 [PSH, ACK] Seq=169 Ack=1 Win=8442 Len=1
797	25.188935	127.0.0.1	127.0.0.1	TCP	44	56449 + 56450 [ACK] Seq=1 Ack=170 Win=8321 Len=0
798	25.189056	127.0.0.1	127.0.0.1	TCP	45	56450 + 56449 [PSH, ACK] Seq=170 Ack=1 Win=8442 Len=1
799	25.189097	127.0.0.1	127.0.0.1	TCP	44	56449 + 56450 [ACK] Seq=1 Ack=171 Win=8321 Len=0
800	25.189557	127.0.0.1	127.0.0.1	TCP	45	49731 + 49732 [PSH, ACK] Seq=170 Ack=1 Win=8442 Len=1
801	25.189577	127.0.0.1	127.0.0.1	TCP	44	49732 + 49731 [ACK] Seq=1 Ack=171 Win=8258 Len=0
802	25.190231	127.0.0.1	127.0.0.1	TCP	45	56450 + 56449 [PSH, ACK] Seq=171 Ack=1 Win=8442 Len=1
> Frame 789: 35 bytes on wire (280 bits), 35 bytes captured (280 bits) on interface \Device\NPF_Loopback, 0x0000 02 00 00 45 00 00 01 f5 15 93 00 00 02 11 00 00 ... E... > Null/Loopback > Internet Protocol Version 4, Src: 172.100.92.160, Dst: 224.1.1.1 > User Datagram Protocol, Src Port: 60338, Dst Port: 5004 > Data (3 bytes)	0010	ac 64 5c a0 e0 01 01 eb b2 13 8c 00 06 4f 18 d\..... 0020	62 79 65 bye			

## CONCLUSIONES

Navarrete Becerril Sharon Anette:

**Versatilidad y adaptabilidad en redes dinámicas:** La implementación de servidores y clientes que utilizan multidifusión ofrece flexibilidad en el manejo de datos distribuidos en redes dinámicas. La lógica basada en bucles infinitos que escuchan mensajes en tiempo real es efectiva para aplicaciones donde la recepción de información continua es crucial, como videoconferencias o sistemas de monitoreo en tiempo real.

## BIBLIOGRAFÍA

Stevens, W. Richard. Programación en red con Unix: La API de Sockets. Addison-Wesley Professional, 2003.

Forouzan, Behrouz A. Comunicaciones de Datos y Redes. McGraw-Hill, 2012. Tanenbaum, Andrew S., y David J. Wetherall. Redes de Computadoras. Pearson, 2010.

Comer, Douglas E. Internetworking con TCP/IP Volumen Uno. Prentice Hall, 2006. Kurose, James F., y Keith W. Ross. Redes de Computadoras: Un Enfoque Descendente. Pearson, 2017.

Beazley, David, y Brian K. Jones. Python Cookbook: Recetas para Dominar Python 3. O'Reilly Media, 2013. Begg, A. Sockets TCP/IP en C: Guía Práctica para Programadores. Morgan Kaufmann, 2000