ING. EN SISTEMAS COMPUTACIONALES

APLICACIONES PARA COMUNICACIONES EN RED

“PRACTICA 2: Envío y recibimiento de broadcast UDP”

PROFESOR: RANGEL GONZALEZ JOSUE

Alumno:

ALMANZA MARTÍNEZ FRANCISCO ALEJANDRO

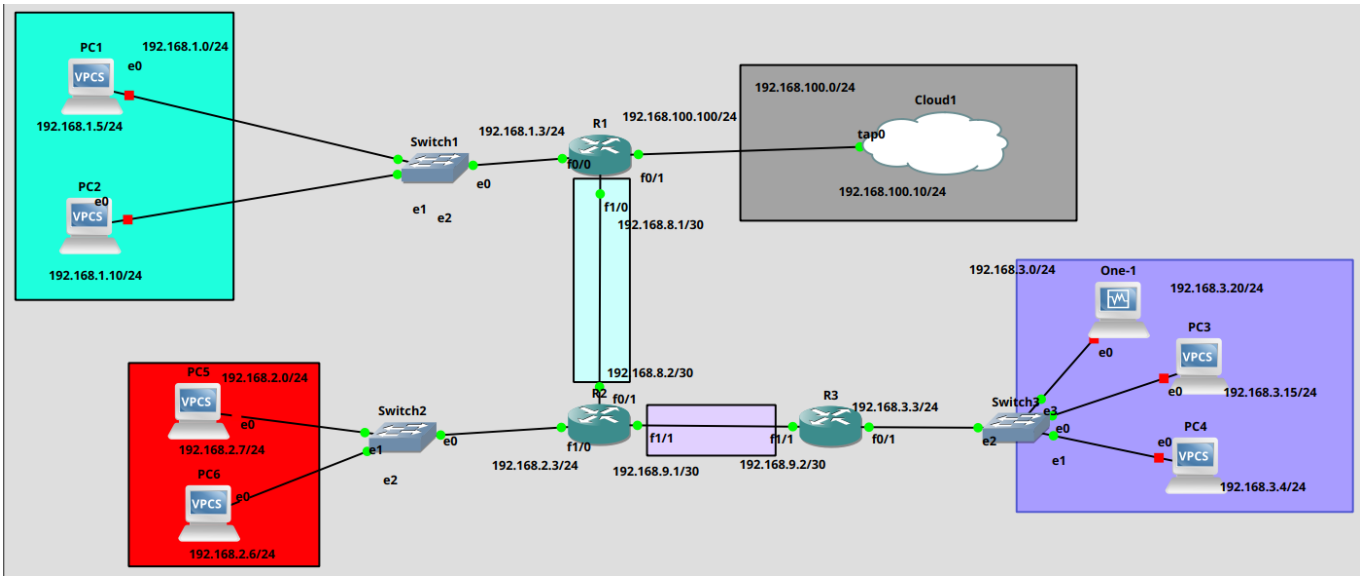
GRUPO: 3CV16

Manual de usuario para realizar el envío y recibimiento a broadcast UDP

Para lograr el procedimiento de este manual se tiene en cuenta los siguiente:

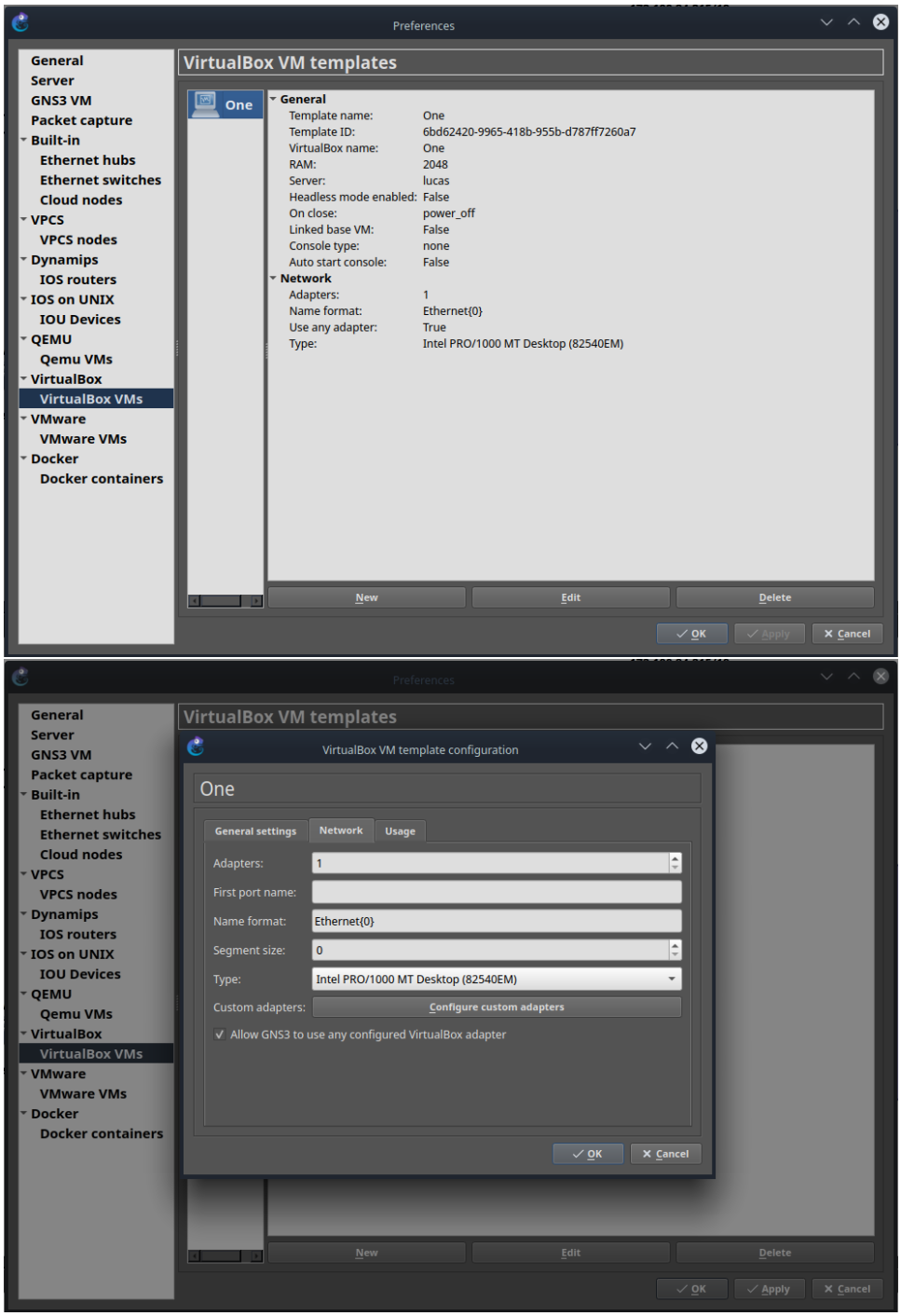
- → El usuario tiene instalado correctamente el simulador de redes GNS3
- → Se puede conectar exitosamente VP's y routers
- → Se tiene instalado correctamente VirtualBox o algún programa de virtualización de máquinas

Se tiene la siguiente topología:

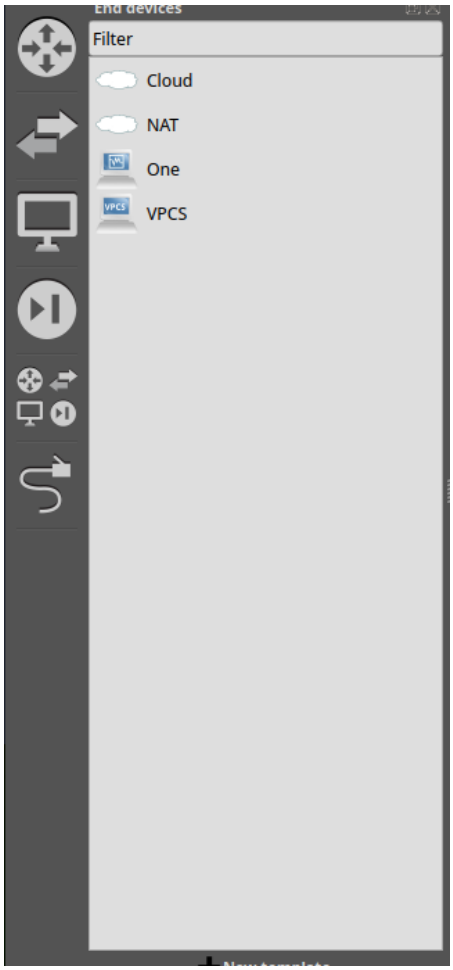


El procedimiento es el siguiente:

1. Una vez configurada y teniendo todas las VCP's máquinas corriendo, seguimos con la configuración de la máquina virtual de Linux
2. Para lo siguiente necesitamos tener instalado "Virtual Box". Agregamos la máquina virtual creada con VirtualBox como plantilla a GNS3
    a. Vamos al menú Editar>Preferencias
    b. Seleccionamos VirtualBox
    c. Agregamos la plantilla
    d. Hacemos clic en editar, vamos a la pestaña de Network y hacemos clic en el checkbox para habilitar la opción de "permitir a GNS3 conectar la máquina virtual con cualquier interface seleccionada"



e. Vamos al menú lateral izquierdo y en la sección de máquina arrastramos la nueva opción cargada de la plantilla de la máquina virtual que acabamos de añadir



f. Conectamos la máquina al switch designado

3. Continuamos con configurar el linux que tenemos en la máquina virtual, así que abrimos la consola y tecleamos los siguientes comandos:
   a. Para saber el dispositivo de la máquina virtual:
      ```
      $ ip addr
      ```
      en este caso es enp0s3. Así que lo copiamos
   b. Para añadir la ip al dispositivo
      ```
      $ sudo ip addres add 192.168.3.20/24 dev enp0s3
      ```
   c. Configurar el dispositivo como disponible
      ```
      $ sudo ip link set dev enp0s3 up
      ```
   d. Y añadimos las siguientes direcciones a ip route especificando que la red está ya conectada
      ```
      $ sudo ip route add 192.168.3.0/24 via 192.168.3.3 enp0s3 onlink
      $ sudo ip route add 192.168.2.0/24 via 192.168.3.3 enp0s3 onlink
      $ sudo ip route add 192.168.1.0/24 via 192.168.3.3 enp0s3 onlink
      $ sudo ip route add 192.168.8.0/30 via 192.168.3.3 enp0s3 onlink
      $ sudo ip route add 192.168.9.0/30 via 192.168.3.3 enp0s3 onlink
      ```

e. Definimos el gateway y dispositivo predeterminado
   $ route add default 192.168.3.3 enp0s3e
f. Hacemos ping para verificar que hay conexión
   $ ping 172.100.84.215

4. Configuramos una interfaz virtual para conectarla con GNS3 y añadimos la configuración a la nube en GNS3
   $ sudo tunctl -t tap0
5. Añadimos una ip a la interfaz virtual perteneciente a la red de la tarjeta de red
   $ sudo ip address add 172.100.84.215/19 dev tap0
6. Añadimos las redes correspondientes al ip route
   $ sudo ip route add $network_id via $gateway_ip dev tap0 onlink
   $ sudo ip route add 192.168.1.0/24 via $gateway_ip dev tap0 onlink
   $ sudo ip route add 192.168.2.0/24 via $gateway_ip dev tap0 onlink
   $ sudo ip route add 192.168.3.0/24 via $gateway_ip dev tap0 onlink
   $ sudo ip route add 192.168.8.0/30 via $gateway_ip dev tap0 onlink
   $ sudo ip route add 192.168.9.0/30 via $gateway_ip dev tap0 onlink
7. Hacemos ping para verificar la conexión
   $ ping 192.168.3.2
8. Una vez verificada la conexión continuamos con configurar los routers para que permitan el envío de paquetes a broadcast. En cada router ingresamos los siguientes comandos
   R1# enable
   R1# configure terminal
   R1(config)# access-list 100 permit icmp any any
   R1(config)# configure terminal [interfaz gateway]
   En las interfaces del router que son gateway ingresamos:
   R1(config-if)# ip directed-broadcast
9. Una vez configurados todos los routers corremos los siguientes programas hechos en lenguaje c:

server.c - Host

```
#include <stdio.h> /* for printf() and fprintf() */
#include <sys/socket.h> /* for socket() and bind() */
#include <arpa/inet.h> /* for sockaddr_in */
#include <stdlib.h> /* for atoi() and exit() */
#include <string.h> /* for memset() */
#include <unistd.h> /* for close() */
#define PORT 5000
#define BUFFER_SIZE 100
#define EVER 1
#define IP_ADDRESS "192.168.3.255"
void DieWithError(char *errorMessage); /* External error handling function */
int main(int argc, char *argv[]){
int sock; /* Socket */
struct sockaddr_in broadcastAddr; /* Broadcast address */
```

```
char *broadcastIP; /* IP broadcast address */
unsigned short broadcastPort; /* Server port */
char *sendString = (char*)malloc(sizeof(char)*BUFFER_SIZE); /* String to broadcast */
int broadcastPermission; /* Socket opt to set permission to broadcast */
unsigned int sendStringLen; /* Length of string to broadcast */
/* Create socket for sending/receiving datagrams */
if ((sock = socket(PF_INET, SOCK_DGRAM, IPPROTO_UDP)) < 0)
DieWithError("socket() failed");
/* Set socket to allow broadcast */
broadcastPermission = 1;
if (setsockopt(sock, SOL_SOCKET, SO_BROADCAST, (void *) &broadcastPermission,
sizeof(broadcastPermission)) < 0)
DieWithError("setsockopt() failed");
/* Construct local address structure */
memset(&broadcastAddr, 0, sizeof(broadcastAddr)); /* Zero out structure */
broadcastAddr.sin_family = AF_INET; /* Internet address family */
broadcastAddr.sin_addr.s_addr = inet_addr(IP_ADDRESS);/* Broadcast IP address */
broadcastAddr.sin_port = htons(PORT); /* Broadcast port */
strcpy(sendString, "Hi! I'm the server");
sendStringLen = strlen(sendString); /* Find length of sendString */
for (;EVER;){ /* Run forever */
printf("Sending message to broadcast: %s ...\n", IP_ADDRESS);
/* Broadcast sendString in datagram to clients every 3 seconds*/
if (sendto(sock, sendString, sendStringLen, 0, (struct sockaddr *)
&broadcastAddr, sizeof(broadcastAddr)) < 0)
DieWithError("[E] Error to send message");
puts("Done! :)");
sleep(3); /* Avoids flooding the network */
}
/* NOT REACHED */
}
void DieWithError(char *errorMessage){
perror(errorMessage);
exit(0);
}
```

client.c - Guest

```
#include <stdio.h> /* for printf() and fprintf() */
```

```
#include <sys/socket.h> /* for socket(), connect(), sendto(), and recvfrom() */
#include <arpa/inet.h> /* for sockaddr_in and inet_addr() */
#include <stdlib.h> /* for atoi() and exit() */
#include <string.h> /* for memset() */
#include <unistd.h> /* for close() */
#define MAXRECVSTRING 255 /* Longest string to receive */
#define PORT 5000
#define EVER 1
void DieWithError(char *errorMessage); /* External error handling function */

int main(int argc, char *argv[]){
int sock; /* Socket */
struct sockaddr_in broadcastAddr; /* Broadcast Address */
unsigned short broadcastPort; /* Port */
char recvString[MAXRECVSTRING+1]; /* Buffer for received string */
int recvStringLen; /* Length of received string */
/* Create a best-effort datagram socket using UDP */
if ((sock = socket(PF_INET, SOCK_DGRAM, IPPROTO_UDP)) < 0)
DieWithError("socket() failed");
/* Construct bind structure */
memset(&broadcastAddr, 0, sizeof(broadcastAddr)); /* Zero out structure */
broadcastAddr.sin_family = AF_INET; /* Internet address family */
broadcastAddr.sin_addr.s_addr = htonl(INADDR_ANY); /* Any incoming interface */
broadcastAddr.sin_port = htons(PORT); /* Broadcast port */
/* Bind to the broadcast port */
if (bind(sock, (struct sockaddr *) &broadcastAddr, sizeof(broadcastAddr)) < 0)
DieWithError("bind() failed");
for(;EVER;){
/* Receive a single datagram from the server */
if ((recvStringLen = recvfrom(sock, recvString, MAXRECVSTRING, 0, NULL, 0)) < 0)
DieWithError("recvfrom() failed");
recvString[recvStringLen] = '\0';
printf("Received: %s\n", recvString); /* Print the received string */
}
close(sock);
exit(0);
}
void DieWithError(char *errorMessage){
perror(errorMessage);
```

```
exit(0);
}
```

Scripts en bash generados para configuración

```
#! /bin/bash
wlp4s0_ip=`ip addr show wlp4s0 | awk '$1 == "inet" {gsub(/\/.*$/, "", $2); print $2}'`
mask=`ip -o -f inet addr show wlp4s0 | awk '/scope global/ {print $4}' | grep -o '[^/]*$'`
read -p "Enter Gateway ip: " gateway_ip
# gateway_ip=$(/sbin/ip route | awk '/default/ { print $3 }')
network_id=$(/sbin/ip route | awk '/scope/ { print $1 }')
echo "[I] Configuring tap0 with tunctl"
sudo tunctl -d tap0
sudo tunctl -t tap0
echo "[I] Done! :)"
echo "[I] wlps4s0 ip: $wlp4s0_ip"
readarray -d . -t ipstr <<<"$wlp4s0_ip"
node_str="${ipstr[3]}"
node=$((node_str))
if [[ node < 254 ]]
then
node=$((node-1))
else
node=$((node+1))
fi
ipstr="${ipstr[0]}.${ipstr[1]}.${ipstr[2]}.$node"
echo "[I] Configuring ip in tap0: $ipstr/$mask"
sudo ip address add $ipstr/$mask dev tap0
echo "[I] Setting up tap0..."
sudo ip link set dev tap0 up
# ip route add <network_ip_id>/<cidr> via <gateway_ip> dev <network_card_name>
# sudo ip route del 192.168.100.10/32 via 192.168.100.1 dev wlp4s0
# sudo ip addr del <network_ip>/<cidr> dev tap0
echo "[I] Adding $network_id to ip route table with gateway ip: $gateway_ip on dev tap0 *onlink*"
```

```
sudo ip route add $network_id via $gateway_ip dev tap0 onlink
sudo ip route add 192.168.1.0/24 via $gateway_ip dev tap0 onlink
sudo ip route add 192.168.2.0/24 via $gateway_ip dev tap0 onlink
sudo ip route add 192.168.3.0/24 via $gateway_ip dev tap0 onlink
sudo ip route add 192.168.8.0/30 via $gateway_ip dev tap0 onlink
sudo ip route add 192.168.9.0/30 via $gateway_ip dev tap0 onlink
# sudo route add default gw $gateway_ip tap0

echo "[I] Done!"
# ping -I tap0 192.168.1.5 # IMPORTANT Select interface for pingging
```