

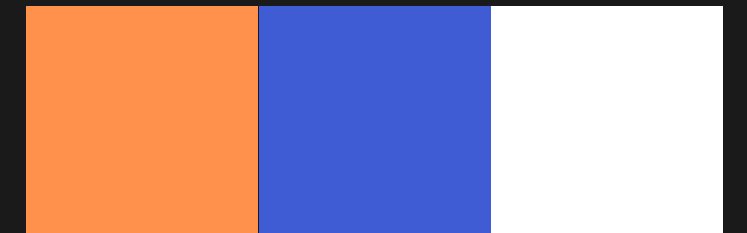
Etapas #4

José Guerra C33510

Sebastián Hernández C23770

Fabrizio Agüero C20097

José Torres C37853



Ciente: Solicitar figura

```
// Cliente de figura normal por TCP
std::string figura = argv[1];
try
{
    Socket client('s');
    std::cout << "[CLIENTE] Conectando al tenedor en " << TENEDOR_IP << ":" << TENEDOR_PORT << "...\\n";
    client.MakeConnection(TENEDOR_IP, TENEDOR_PORT);
    std::cout << "[CLIENTE] Conectado al tenedor.\\n";

    std::string http_request = "GET /figure?name=" + figura + " HTTP/1.1\\r\\nHost: cliente\\r\\n\\r\\n";
    std::cout << "[CLIENTE] Enviando solicitud HTTP: " << http_request;

    client.Write(http_request.c_str(), http_request.size());

    char buffer[BUFFER_SIZE];
    std::string total_response;

    while (true)
    {
        ssize_t bytes = client.Read(buffer, sizeof(buffer) - 1);
        if (bytes <= 0)
            break;
        buffer[bytes] = '\\0';
        total_response += buffer;
    }

    std::cout << "[CLIENTE] Respuesta recibida:\\n"
              << total_response << std::endl;

    client.Close();
}
```

Shutdown

```
for (const auto &ip : broadcast_ips)
{
    inet_pton(AF_INET, ip.c_str(), &addr.sin_addr);
    udp.sendto(mensaje.c_str(), mensaje.size(), &addr);
    std::cout << "[SERVER] Enviado broadcast de apagado a " << ip << ": " << mensaje << "\n";
}
```

Fork: Descubrimiento

```
while (true) {  
    cout << "[DISCOVERY] Enviando solicitudes de descubrimiento\n";  
  
    // Enviar broadcast a todas las IPs  
    string mensaje = "GET /servers";  
    for (const auto &ip : broadcast_ips) {  
        inet_pton(AF_INET, ip.c_str(), &addr.sin_addr);  
        send_sock.sendTo(mensaje.c_str(), mensaje.size(), &addr);  
        cout << "[BROADCAST] Enviado a " << ip << endl;  
    }  
  
    this_thread::sleep_for(chrono::seconds(BROADCAST_WAIT));  
}
```

Fork: Descubrimiento

```
// Manejar anuncios de servidores
else if (mensaje != "GET /servers") { // Ignorar broadcasts de otros tenedores
    istringstream iss(mensaje);
    string nombre, ip, lista;

    if (getline(iss, nombre, '|') &&
        getline(iss, ip, '|') &&
        getline(iss, lista)) {

        // Limpiar solo espacios alrededor del separador
        nombre.erase(nombre.find_last_not_of(' ') + 1);
        nombre.erase(0, nombre.find_first_not_of(' '));
        ip.erase(ip.find_last_not_of(' ') + 1);
        ip.erase(0, ip.find_first_not_of(' '));
        lista.erase(lista.find_last_not_of(' ') + 1);
        lista.erase(0, lista.find_first_not_of(' '));

        // Validar IP
        struct sockaddr_in tmp;
        if (inet_pton(AF_INET, ip.c_str(), &tmp.sin_addr) != 1) {
            cerr << "[UDP] IP inválida recibida: " << ip << endl;
            continue;
        }

        istringstream figs_stream(lista);
        string figura;
        lock_guard<mutex> lock(tabla_mutex);
        while (getline(figs_stream, figura, ',')) {
            // Limpiar nombre de figura
            figura.erase(figura.find_last_not_of(' ') + 1);
            figura.erase(0, figura.find_first_not_of(' '));

            tabla_ruteo[figura] = ip;
            cout << "[RUTEO] Figura '" << figura << "' registrada con IP " << ip << endl;
        }
    }
}
```

Fork: Hilos

```
// -----  
// Hilo unificado para escucha UDP (descubrimiento y shutdown)  
// -----  
void unified_udp_listener() {  
    Socket sock('d');  
    sock.BuildSocket('d');  
  
    // Configurar socket  
    int yes = 1;  
    setsockopt(sock.idSocket, SOL_SOCKET, SO_REUSEADDR, &yes, sizeof(yes));  
    setsockopt(sock.idSocket, SOL_SOCKET, SO_BROADCAST, &yes, sizeof(yes));  
    sock.Bind(DISCOVERY_PORT);  
  
    // Configurar timeout  
    struct timeval timeout;  
    timeout.tv_sec = TIMEOUT_RESPONSE;  
    timeout.tv_usec = 0;  
    setsockopt(sock.idSocket, SOL_SOCKET, SO_RCVTIMEO, &timeout, sizeof(timeout));  
  
    cout << "[UDP] Escuchando en puerto " << DISCOVERY_PORT << " (descubrimiento y shutdown)\n";  
}
```

Fork: Hilos

```
// -----  
// Hilo que envía solicitudes de descubrimiento  
// -----  
void discovery_sender() {  
    Socket send_sock('d');  
    send_sock.BuildSocket('d');  
  
    // Permitir broadcast  
    int yes = 1;  
    setsockopt(send_sock.idSocket, SOL_SOCKET, SO_BROADCAST, &yes, sizeof(yes));  
  
    sockaddr_in addr{};  
    addr.sin_family = AF_INET;  
    addr.sin_port = htons(DISCOVERY_PORT);
```

Fork: Hilos

```
// -----  
// Hilo HTTP que atiende a múltiples clientes  
// -----  
void atender_clientes_http() {  
    VSocket *servidor = new Socket('s');  
    servidor->Bind(CLIENT_HTTP_PORT);  
    servidor->MarkPassive(5);  
    cout << "[HTTP] Servidor escuchando en puerto " << CLIENT_HTTP_PORT << "\n";  
  
    while (true) {  
        VSocket *cliente = servidor->AcceptConnection();  
        thread(manejar_peticion_http, cliente).detach();  
    }  
  
    delete servidor;  
}
```


Fork: Hilos

```
// -----  
// Función principal  
// -----  
int main() {  
    cout << "[INIT] Tenedor de figuras iniciado\n";  
    cout << " - Puerto HTTP: " << CLIENT_HTTP_PORT << "\n";  
    cout << " - Puerto descubrimiento: " << DISCOVERY_PORT << "\n";  
  
    // Iniciar hilos  
    thread t1(unified_udp_listener); // Escucha UDP unificada  
    thread t2(discovery_sender);    // Envía broadcasts de descubrimiento  
    thread t3(atender_clientes_http); // Servidor HTTP  
  
    t1.join();  
    t2.join();  
    t3.join();  
  
    return 0;  
}
```

Servidor: Figuras

```
while (true) {
    size_t len = recv_sock.recvFrom(buffer, sizeof(buffer) - 1, &cliente);
    buffer[len] = '\0';
    string mensaje(buffer);

    if (mensaje == "GET /servers") {
        vector<string> figs = fs.get_figuras();
        ostringstream oss;
        for (size_t i = 0; i < figs.size(); ++i) {
            oss << figs[i];
            if (i != figs.size() - 1)
                oss << ",";
        }

        string respuesta = string(SERVER_NAME) + " | " + SERVER_IP + " | " + oss.str();
        send_sock.sendTo(respuesta.c_str(), respuesta.size(), &cliente);

        cout << "[DISCOVERY] Respondió: " << respuesta << endl;
    }
    else if (mensaje.rfind("Shutdown", 0) == 0) {
        cout << "[INFO] Mensaje de apagado recibido, ignorado: " << mensaje << endl;
    }
    else {
        cout << "[UDP] Mensaje ignorado: " << mensaje << endl;
    }
}
```

Servidor: Figuras

```
// se necesita llamar a find_free_block() para obtener un bloque libre
// además de escribir el bloque de datos en la posición correspondiente
short bloque_libre = this->find_free_block();
this->escribir_bloque(bloque_libre, datos); // Escribir los datos en el bloque libre
*(short*)&entrada[26] = bloque_libre; // Supuesto bloque de datos
```

Servidor: Hilos

```
// -----  
// Hilo que responde solicitudes de descubrimiento (UDP)  
// -----  
void discovery_listener() {  
    Socket recv_sock('d');  
    recv_sock.BuildSocket('d');  
  
    // Configurar para reutilizar dirección  
    int yes = 1;  
    setsockopt(recv_sock.idSocket, SOL_SOCKET, SO_REUSEADDR, &yes, sizeof(yes));  
  
    // Bind al puerto de descubrimiento  
    recv_sock.Bind(SERVER_DISCOVERY_PORT);  
  
    // Socket para enviar respuestas (no necesita bind)  
    Socket send_sock('d');  
    send_sock.BuildSocket('d');  
  
    sockaddr_in cliente{};  
    char buffer[512];  
  
    cout << "[UDP] Escuchando descubrimiento en puerto " << SERVER_DISCOVERY_PORT << "...\\n";
```

Servidor: Hilos

```
// -----  
// hilo que anuncia el servidor activamente por broadcast  
// -----  
void broadcast_advertiser()  
{  
    Socket s('d');  
    s.BuildSocket('d');  
  
    int yes = 1;  
    setsockopt(s.idSocket, SOL_SOCKET, SO_BROADCAST, &yes, sizeof(yes));  
  
    sockaddr_in addr{};  
    addr.sin_family = AF_INET;  
    addr.sin_port = htons(SERVER_DISCOVERY_PORT);
```

Servidor: Hilos

```
// -----  
// Hilo TCP que responde con la figura ASCII  
// -----  
void tcp_figure_server()  
{  
    VSocket *servidor = new Socket('s');  
    servidor->Bind(SERVER_PORT);  
    servidor->MarkPassive(5);  
  
    cout << "[TCP] Servidor escuchando en puerto " << SERVER_PORT << "\n";  
  
    while (true)  
    {  
        VSocket *cliente = servidor->AcceptConnection();  
  
        char buffer[512] = {0};  
        cliente->Read(buffer, sizeof(buffer) - 1);  
        buffer[511] = '\0';  
  
        string request(buffer);  
        string prefix = "GET /figure/";  
        size_t pos = request.find(prefix);
```

Servidor: Hilos

```
// -----  
// MAIN con hilos  
// -----  
int main()  
{  
    thread t1(discovery_listener);  
    thread t2(tcp_figure_server);  
    thread t3(broadcast_advertiser);  
  
    cout << "[INIT] Servidor de figuras '" << SERVER_NAME << "' activo\n";  
  
    t1.join();  
    t2.join();  
    t3.join();  
  
    return 0;  
}
```

Protocolo grupal

Protocolo de comunicación Tenedor a Server Grupo 03

Tenedor se conecta al Servidor (Nuevo tenedor):

Tenedor hace broadcast del siguiente mensaje:

Request: GET /servers

Los servidores contestan directamente al tenedor que lo solicitó:

Response: ServerName | ip | {lista dibujos}

Ejemplo de respuesta del servidor:

ServidorA 172.16.123.51 ballena, perro, gato
--

- El **tenedor guarda la información localmente**, para luego hacer solicitudes de ascii-arts al servidor que corresponda.
- Al ser un mensaje de broadcast y los tenedores estar en la misma red estos recibirán las peticiones **GET /servers**, las cuales debe ignorar.

Protocolo grupal

Servidor se conecta al Tenedor:

Los servidores envían a los tenedores en broadcast:

Response: `ServerName | ip | {lista dibujos}`

Ejemplo:

ServidorA | 172.16.123.51 | ballena, perro, gato

- Al ser un mensaje de broadcast y los servidores estar en la misma red estos recibirán `ServerName | ip | {lista dibujos}`, las cuales debe ignorar.

Protocolo grupal

Petición ASCII-art tenedor a servidor:

Request: GET /figure/{nombre_figura}

Respuesta del servidor:

Si la figura existe: (Se envía solo el ASCII-art)

```
(\__/\)\n(='!')\n(\")_(\")
```

Si la figura no existe: el Tenedor tiene la tabla de figuras actualizada por lo que no hará la solicitud a ningún Servidor y retorna un código de error: 404 Not Found

```
404 Not Found
```

Protocolo grupal

Servidor se conecta al Tenedor:

Los servidores envían a los tenedores en broadcast:

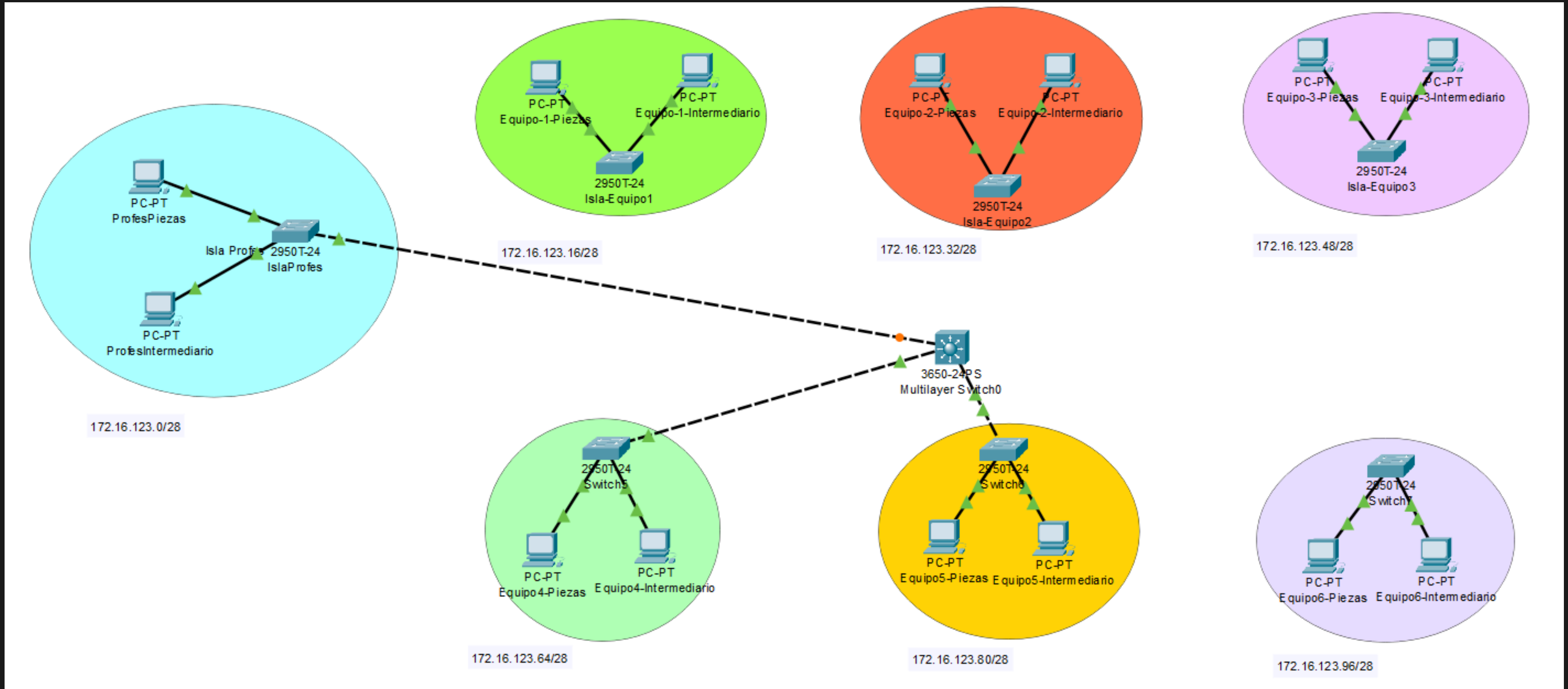
Response: `ServerName | ip | {lista dibujos}`

Ejemplo:

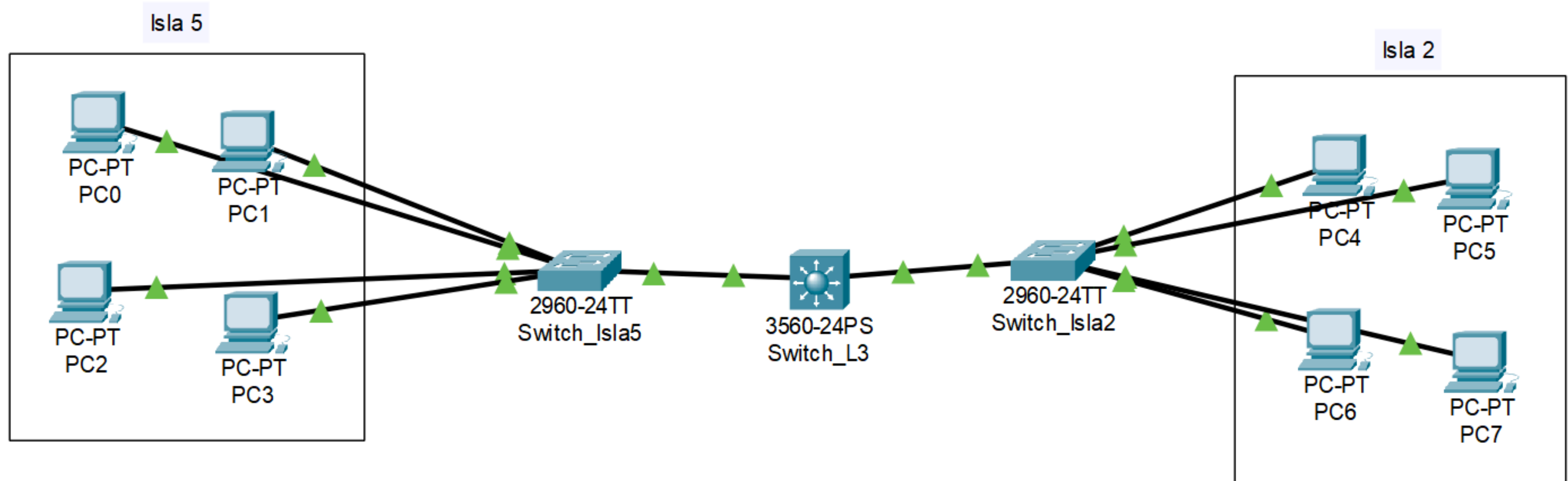
ServidorA 172.16.123.51 ballena, perro, gato
--

- Al ser un mensaje de broadcast y los servidores estar en la misma red estos recibirán `ServerName | ip | {lista dibujos}`, las cuales debe ignorar.

Packet Tracer



Packet Tracer



Configuración de los switches

Manual de Configuración: Switch Cisco Catalyst 2960 – Isla 5 (VLAN 350)

Paso 1: Borrar la configuración anterior y reiniciar el switch

Entrar al modo privilegiado:

```
Switch> enable
```

Borrar la configuración guardada (startup-config) y VLANs anteriores:

```
Switch# delete flash:config.text
```

Delete filename [config.text]? [ENTER] %Error deleting flash:config.text (No such file or directory) <-- Si no existe, es normal.

```
Switch# delete flash:vlan.dat
```

Delete filename [vlan.dat]? [ENTER] Delete flash:vlan.dat? [confirm] [ENTER]

Reiniciar el switch:

```
Switch# reload
```

Proceed with reload? [confirm] [ENTER] Al reiniciar, salir del diálogo de configuración inicial:

Would you like to enter the initial configuration dialog? [yes/no]: no