Entregable Trabajo Práctico N° 4

Tomás Vidal
Sistemas Operativos y Redes
Facultad de Ingeniería, UNLP, La Plata, Argentina.
28 de Noviembre, 2024.

I. Redes presentadas

En la figura 1 se muestra la topología de las redes que fueron asignadas. Para hacer el análisis de la misma se empleó la dirección IP asignada *181.29.152.0* con CIDR[1] *22*.

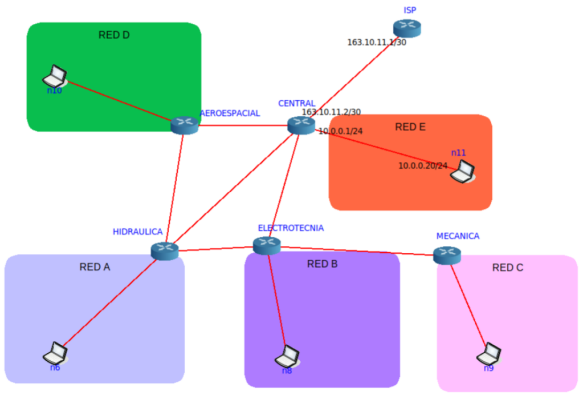

Fig. 1. Topología de la red dada

I-A. Dirección IP dada

La dirección de IP dada es *181.29.148.0/22*, lo que implica que se pueden asignar un total de **1024 hosts**. Como la IP comienza con la dirección *181*, la misma pertenece a la **clase B**

I-B. Asignación de IPs

Para hacer la asignación de IPs a las diferentes interfaces y dispositivos se hizo *subnetting*[2]. El proceso empleado consistió en *dividir recursivamente* la IP dada en subredes más pequeñas, teniendo en cuenta las redes con mayor requerimiento de *hosts*[3], es decir la **Red B**, luego la **Red A**, **D**, **C** y por último los enlaces punto a punto, para los cuales se empleó un CIDR de 30 (lo que permite optimizar la cantidad hosts).

[1]El enrutamiento entre dominios sin clases (CIDR) es un método de asignación de direcciones IP que mejora la eficiencia del enrutamiento de datos en Internet

[2]Subnetting es el proceso de dividir una red IP más grande en subredes más pequeñas, con el objetivo de optimizar el uso de direcciones IP, mejorar la seguridad y gestionar el tráfico de red de manera más eficiente.

[3]Hosts son dispositivos conectados a una red que pueden enviar y recibir datos, como computadoras, servidores, teléfonos inteligentes, impresoras o cualquier otro equipo con una dirección IP asignada.

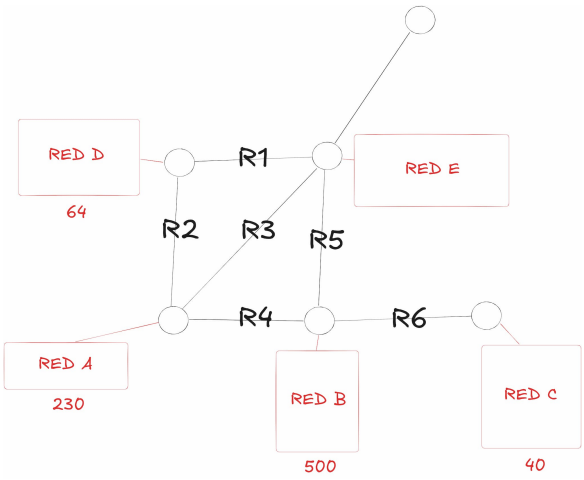

Fig. 2. Diseño de la red para hacer subnetting

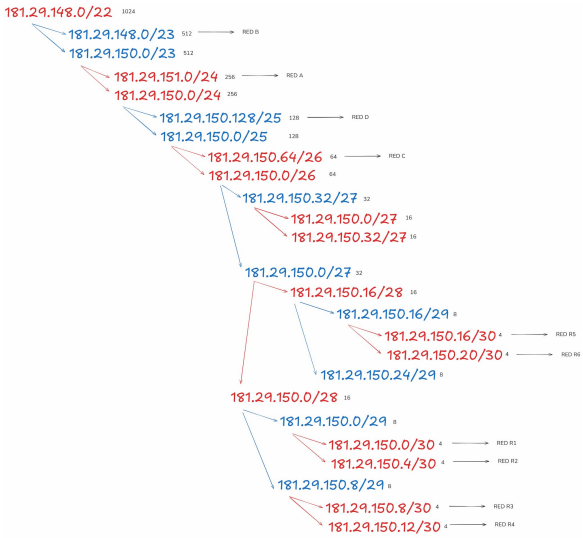

Fig. 3. Diseño subnetting de la red

I-C. Simulación en Core Emulator

Luego de diseñar la red y las subredes se aplicaron las IPs en el simulador Core Emulator, y para que se puedan transferir paquetes entre estas redes correctamente se hicieron enrutamientos en los diferentes *routers* con los comandos:

1. ip route add [GATEWAY *cual*]/[CIDR] via [IP *desde*]

2. ip route del [GATEWAY *cual*]/[CIDR] via [IP *desde*]

El comando 1 se empleó para agregar redes (revelar tablas de enrutamiento), y el comando 2 para eliminar una red que era incorrecta. Para guardar los cambios hechos en las tablas de enrutamiento se empleó el script provisto *save*, el cual salva los cambios en el archivo "core.txt", además se puede emplear el comando *restore core.txt* para restaurar estos cambios cuando se recarga el archivo de la topología en el Core Emulator.

Se corroboró el correcto enrutamiento de paquetes entre las redes por medio del comando *traceroute*, proporcionado en una interfaz gráfica en Core Emulator, los resultados se muestran a continuación:

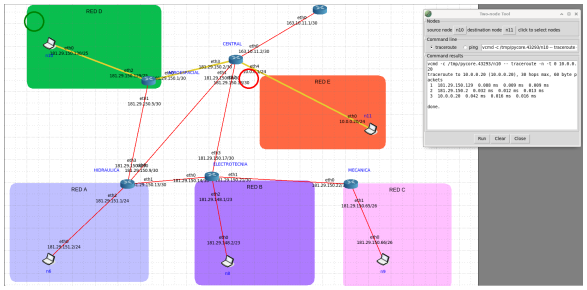

Fig. 4. Traceroute desde n10 a n11

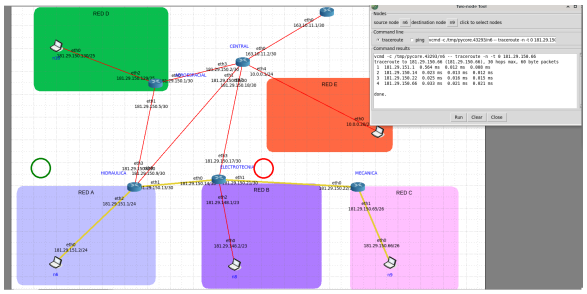

Fig. 5. Traceroute desde n6 a n9

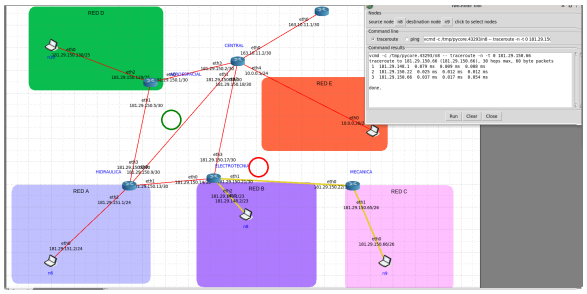

Fig. 6. Traceroute desde n8 a n9

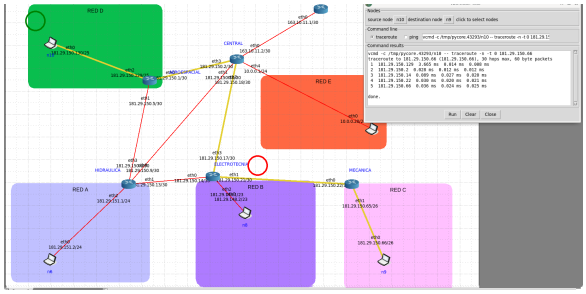

Fig. 7. Traceroute desde n10 a n9

Para corroborar que los dispositivos tuvieran conexión a internet se hizo *traceroute* de los diferentes dispositivos a la IP *163.10.11.1* (ISP).

En la realidad habría que hacer un ping o traceroute a la dirección 8.8.8.8 por ejemplo, o alguna conocida como google.com.ar. Pero en este caso se simula que la conexión a internet está hecha cuando se pueden dirigir paquetes al ISP.

También sería necesario implementar un NAT[4] en el *router central* o en el *router del ISP* para enmascarar las IPs privadas. Sin embargo, si se tuviera que implementar en un escenario real, no sería posible hacerlo en el *router del ISP* debido a la falta de acceso, por lo que únicamente podría realizarse en el *router central*.

II. Implementación del servidor TCP

El protocolo TCP descompone los datos en paquetes y los reenvía a la capa del protocolo de Internet (IP) para garantizar que cada mensaje llegue a su ordenador de destino. El estado actual de desarrollo del protocolo TCP permite establecer una conexión entre dos puntos terminales en una red informática común que posibilite un intercambio mutuo de datos. En este proceso, cualquier pérdida de datos se detecta y resuelve, por lo que se considera un protocolo fiable. La secuencia específica para establecer una conexión con el protocolo TCP es la siguiente:

1. En el primer paso, el cliente que desea establecer la conexión envía al servidor un paquete SYN o segmento SYN (del inglés synchronize = "sincronizar") con un número de secuencia individual y aleatorio. Este número garantiza la transmisión completa en el orden correcto (sin duplicados).
2. Si el servidor ha recibido el segmento, confirma el establecimiento de la conexión mediante el envío de un paquete SYN-ACK (del inglés acknowledgement = "confirmación") incluido el número de secuencia del cliente después de sumarle 1. De forma adicional, transmite un número de secuencia propio al cliente.
3. Para finalizar, el cliente confirma la recepción del segmento SYN-ACK mediante el envío de un paquete

[4]NAT (Network Address Translation) es una técnica utilizada en redes para modificar las direcciones IP en los paquetes de datos que pasan a través de un router o firewall. Permite que múltiples dispositivos en una red local (LAN) compartan una única dirección IP pública, facilitando la conservación de direcciones IPv4 y añadiendo una capa de seguridad al ocultar las direcciones internas de los dispositivos.

ACK propio, que en este caso cuenta con el número de secuencia del servidor después de sumarle 1. En este punto también puede transmitir ya los primeros datos al servidor.

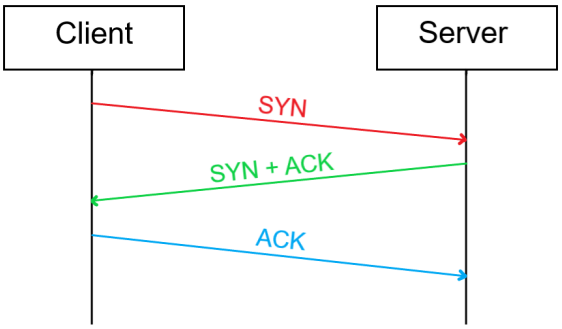

Fig. 8. Diagrama de la sincronización del TCP

II-A. *Cabecera TCP*

Bits	0-15			16-31
0	Source port			Destination port
32	Sequence number			
64	Acknowledgment number			
96	Offset	Reserved	Flags	Window size
128	Checksum			Urgent pointer
160	Options			

Fig. 9. Estructura de la cabecera TCP

II-B. *Lógica del código*

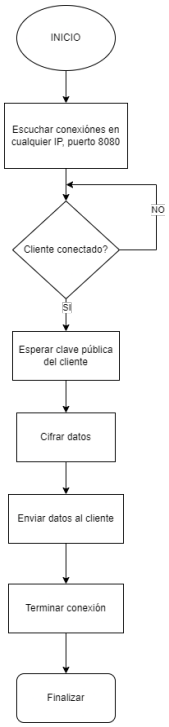

Fig. 10. Diagrama de flujo del servidor

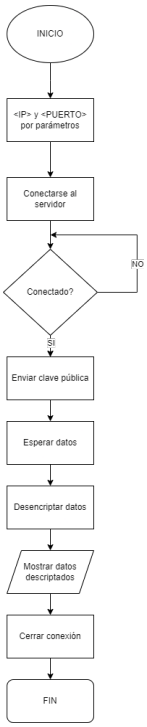

Fig. 11. Diagrama de flujo del cliente

Los diagramas 11 y 10 explican el comportamiente general de ambos programas. Las principales características son:

II-B.1. *Cliente:*

- Debe recibir como parámetro la **IP** y el **puerto** del servidor para conectarse.
- Debe **enviar** la clave pública de encriptación.
- Imprime los datos desencriptados en pantalla.

II-B.2. *Server:*

- El puerto por defecto es **8080**.
- Debe **recibir** la clave pública de encriptación.
- Enviar los datos encriptados con la llave pública.

En los códigos se pueden habilitar los comentarios de debugeo descomentando el macro DEBUG. Para recompilar los códigos se puede emplear el Makefile, corriendo: make client y make server.

II-C. *Capturas de paquetes con Wireshark*

Se ejecutó el servidor en el dispositvo **n6** (*181.29.150.2*) y el cliente en el dispositvo **n10** (*181.29.150.130*), y con Wireshark[5] se hizo la captura de los paquetes en el cliente. En las siguientes figuras se muestra como se capturan los datos, además se puede ver como se hace la conexión TCP como se explicó en 8.

[5]Wireshark es un analizador de protocolos de red de código abierto que permite capturar y examinar en detalle el tráfico de datos en una red en tiempo real. Es ampliamente utilizado en la industria para diagnóstico de problemas, desarrollo de software y aprendizaje de protocolos de comunicación.

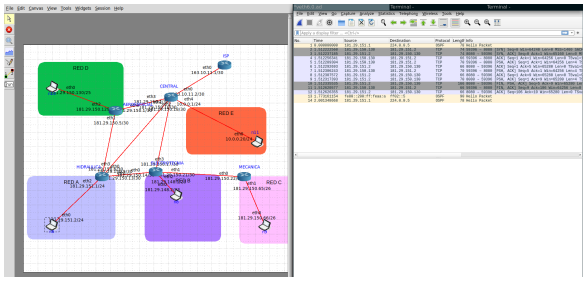

Fig. 12. Trafico entre **n6** y **n10**

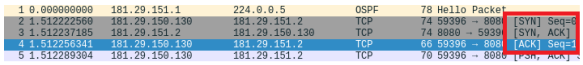

Fig. 13. Trafico entre **n6** y **n10**, TCP

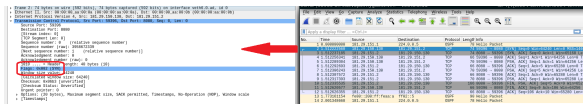

Fig. 14. Trafico entre **n6** y **n10**, TCP SYN

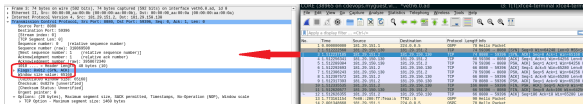

Fig. 15. Trafico entre **n6** y **n10**, TCP SYN-ACK

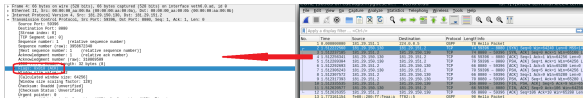

Fig. 16. Trafico entre **n6** y **n10**, TCP ACK

A continuación, se muestran los paquetes que contienen los datos. En la figura 17 se observa que la bandera PUSH está en 1, lo que indica que la transmisión se realiza de un dato (número) a la vez. También se puede notar que la transmisión es del cliente al servidor, ya que dicho dato corresponde a la clave pública. Posteriormente, en la figura 18 se muestra cómo el TCP Segment Len es distinto de 0, ya que se están transmitiendo datos. Este paquete en particular corresponde a uno de los números encriptados que el servidor transmite.

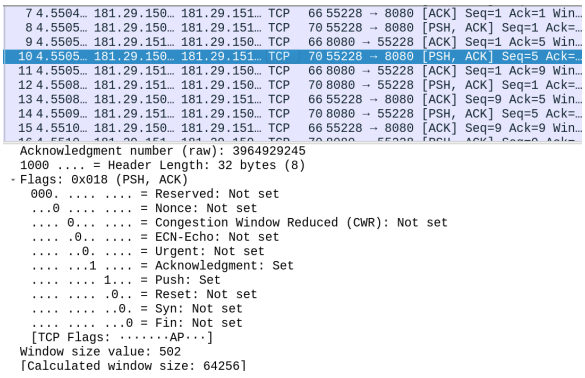

Fig. 17. Trafico entre **n6** y **n10**

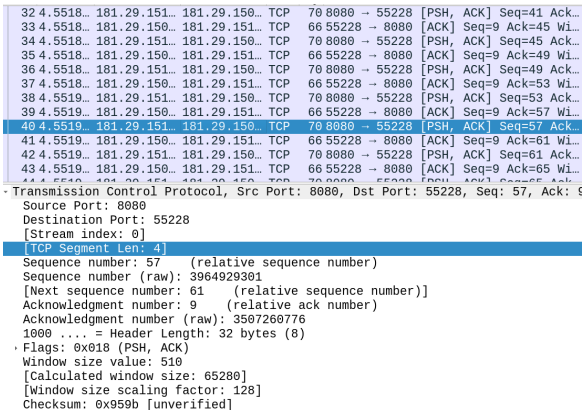

Fig. 18. Trafico entre **n6** y **n10**

II-D. Archivos

Junto al presente se adjuntan las imágenes para que el lector las pueda ver con más detalle, además se encuentra un archivo (“core.txt”) que contiene los datos sobre el enrutamiento de la topología dada.