Solución Examen – 17 de febrero de 2025
(ref: sol_erc202502.odt)

Instrucciones

- **Indique su nombre completo y número de cédula en cada hoja.**
- **Numere todas las hojas e indique la cantidad total de hojas que entrega en la primera.**
- **Escriba las hojas de un solo lado y utilice una caligrafía claramente legible.**
- **Comience cada pregunta teórica y cada ejercicio en una hoja nueva.**
- **Solo se responderán dudas de letra. No se responderán dudas de ningún tipo los últimos 30 minutos del examen.**
- **El examen es individual y sin material. Apague su teléfono celular mientras esté en el salón del examen.**
- **Es obligatorio responder correctamente al menos 15 puntos en las preguntas teóricas y 20 de los problemas prácticos. Los puntos ganados en el curso se suman a los puntos de teórico.**
- **El puntaje mínimo de aprobación es de 60 puntos.**
- **Para todos los ejercicios, si es necesario, puede suponer que dispone de los tipos de datos básicos (p.ej. lista, cola, archivo, string, etc.) y sus funciones asociadas (ej: tail(lista), crear(archivo), concatenar(string, string).**
- **Justifique todas sus respuestas.**
- **Duración: 3 horas. Culminadas las 3 horas el alumno no podrá modificar las hojas a entregar de ninguna forma.**

Preguntas Teóricas

Pregunta 1 (10 puntos)

a) Mencione y describa brevemente los 5 métodos de HTTP vistos en el curso.
b) ¿Cómo se especifica el método a utilizar en una comunicación HTTP?
c) Indique cuáles de ellos no están considerados en HTTP 1.0.

Solución

a)
En HTTP los métodos son la forma de indicar qué se quiere hacer (acción) con determinado recurso.
Los 5 métodos vistos en el curso son: GET, POST, HEAD, PUT y DELETE.
El método GET se emplea por ejemplo cuando un navegador solicita un objeto, identificándose dicho objeto en el campo URL.
Con un mensaje POST, por ejemplo el usuario solicita también una página web al servidor, pero el contenido concreto de la misma dependerá de lo que el usuario haya escrito en el o los campos de un formulario.
Cuando un servidor recibe una solicitud con el método HEAD, responde con un mensaje HTTP, pero excluye el objeto solicitado (debugging).
El método PUT suele utilizarse junto con herramientas de publicación web. Esto permite a un usuario cargar un objeto en una ruta específica (directorio) en un servidor web determinado.
El método DELETE permite a un usuario o a una aplicación borrar un objeto de un servidor web.
b)
El método de una solicitud HTTP se especifica en la primera línea de la cabecera, donde se identifican 3 campos: el campo de método, el campo URL y el campo de la versión HTTP.
El campo que especifica el método puede tomar diferentes valores, entre los que se incluyen GET, POST, HEAD, PUT y DELETE (vistos en la parte a).
c)
En HTTP 1.0 no están especificados los métodos PUT y DELETE.

Pregunta 2 (10 puntos)

a) Explique por qué se dice que TCP intenta ser un protocolo justo (fair).
b) ¿Que se podría hacer, desde el lugar de un desarrollador de una aplicación que utiliza TCP, para intentar obtener un mayor ancho de banda para dicha aplicación?
c) ¿Por qué UDP no es un protocolo fair?

Solución

a) Si suponemos K conexiones TCP (lo suficientemente duraderas) que atraviesan un enlace cuello de botella cuyo ancho de banda R bps es menor a la suma de los anchos de banda de las K conexiones individuales, y además suponemos que que no existe tráfico UDP en la red, entonces los algoritmos de control de congestión de TCP llevarán a que la tasa de transmisión de cada conexión individual tienda a R/K. A este fenómeno se le denomina fairness.

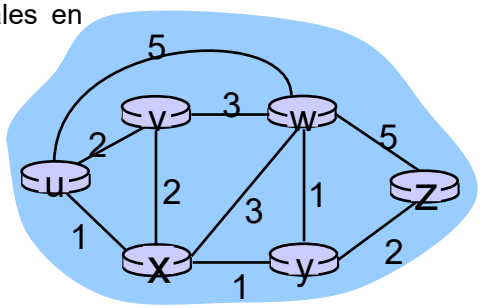
b) Se podría implementar la aplicación de tal forma que abriera conexiones TCP concurrentes, para lograr un mayor ancho de banda en total para la aplicación, aplicando el concepto de fairness en forma egoísta.

c) Dado que UDP no tiene control de congestión, puede enviar segmentos a la velocidad que le permita el medio (o le demande la aplicación) y no se auto-limita acorde al estado de la red. De esta forma, UDP podría consumir un mayor ancho de banda frente a las conexiones TCP que coexistan en el mismo tiempo.

Pregunta 3 (10 puntos)

Dada la red de la figura, ejecute paso a paso el algoritmo de Dijkstra para encontrar los caminos más cortos desde el origen **u** hacia todos los nodos.

Justifique textualmente cada paso, mostrando los resultados parciales en una tabla de la forma:



Paso	N'	D(v),p(v)	D(w),p(w)	D(x),p(x)	D(y),p(y)	D(z),p(z)

Obs: agregue todas las filas necesarias.

Solución

Paso	N'	D(v),p(v)	D(w),p(w)	D(x),p(x)	D(y),p(y)	D(z),p(z)
0	u	2,u	5,u	1,u	∞	∞
1	ux	2,u	4,x		2,x	∞
2	uxy	2,u	3,y			4,y
3	uxyv		3,y			4,y
4	uxyvw					4,y
5	uxyvwz					

En el paso de inicialización del algoritmo, las rutas de menor costo conocidas actualmente desde **u** hasta sus vecinos directamente conectados, **v**, **x** y **w**, se inicializan en 2, 1 y 5, respectivamente. Nótese en particular que el costo para **w** se establece en 5 (aunque pronto veremos que existe una ruta de menor costo) ya que este es el costo del enlace directo (un salto) desde **u** hasta **w**. Los costos para **y** y **z** se establecen en infinito porque no están conectados directamente a **u**.

En la primera iteración, buscamos entre aquellos nodos que aún no se han agregado al conjunto N′ y encontramos el nodo con el menor costo al final de la iteración anterior. Ese nodo es **x**, con un costo de 1, y por lo tanto **x** se agrega al conjunto **N′**. Ejecutamos entonces la actualización **D(i)** para todos los nodos **i**, lo que produce los resultados que se muestran en la segunda línea (paso 1) de la Tabla.

recordar que D(i) = min(D(i), D(j)+ c(j,i))

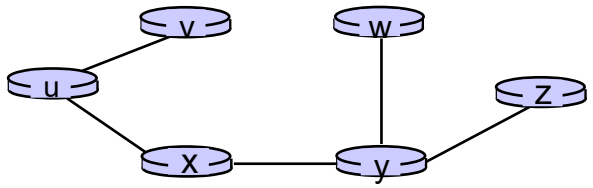
El costo de la ruta a **v** no cambia. El costo de la ruta a **w** (que era 5 al final de la inicialización) a través del nodo **x** tiene un costo de 4. Por lo tanto, se selecciona esta ruta de menor costo y el predecesor de

w a lo largo de la ruta más corta desde **u** se establece en **x**. De manera similar, el costo de **y** (a través de **x**) se calcula como 2, y la tabla se actualiza en consecuencia.

En la segunda iteración, se descubre que los nodos **v** e **y** tienen las rutas de menor costo (2), y rompemos el empate arbitrariamente y agregamos **y** al conjunto **N′** de modo que **N′** ahora contiene **u**, **x** e **y**. El costo para los nodos restantes que aún no están en **N'**, es decir, los nodos **v**, **w** y **z**, se actualizan como indica el algoritmo para **D(i)**, lo que produce los resultados que se muestran en la tercera fila de la Tabla.

Y así sucesivamente...

Cuando el algoritmo LS termina, tenemos, para cada nodo, su predecesor a lo largo de la ruta de menor costo desde el nodo de origen. Para cada predecesor, también tenemos su predecesor, y de esta manera podemos construir la ruta completa desde el origen hasta todos los destinos. La tabla de reenvío en el nodo **u**, se puede construir a partir de esta información almacenando, para cada destino, el nodo del siguiente salto en la ruta de menor costo desde **u** hasta el destino. En la siguiente Figura se muestran las rutas de menor costo resultantes y la tabla de reenvío de **u**.



Destino	Enlace de salida
v	(u,v)
w	(u,x)
x	(u,x)
y	(u,x)
z	(u,x)

Pregunta 4 (10 puntos)

a) Describa brevemente los mecanismos de detección/corrección de errores de UDP, IP y Ethernet considerados en el curso.
b) Dada una comunicación UDP en una red IP y donde todos los segmentos de red involucrados en la misma utilizan la tecnología Ethernet, ¿es necesario contar con funcionalidades de gestión de errores en las 3 capas? Justifique su respuesta.

Solución

a) UDP provee un mecanismo de detección de errores mediante la incorporación al encabezado de cada segmento de una suma de comprobación (checksum). UDP en el lado del emisor calcula el complemento a 1 de la suma de todas las palabras de 16 bits del segmento, acarreando cualquier desbordamiento obtenido durante la operación de suma sobre el bit de menor peso. Este resultado se almacena en el campo suma de comprobación del segmento. Si se detecta error, se descarta el segmento.
En el caso de IP, la suma de comprobación de cabecera ayuda a los routers a detectar errores de bit en el encabezado de un datagrama IP recibido. Esta suma de comprobación se calcula tratando cada pareja de 2 bytes de la cabecera como un número y sumando dichos números utilizando aritmética de complemento a 1. El complemento a 1 de esta suma, conocido como suma de comprobación Internet, se almacena en el campo Suma de comprobación. Un router calcula la suma de comprobación de cabecera para cada datagrama IP recibido y detecta una condición de error si la suma de comprobación incluida en la cabecera del datagrama no coincide con la suma de comprobación calculada. Normalmente, los routers descartan los datagramas en los que se ha detectado que existe un error. Observe que la suma de comprobación tiene que volver a calcularse y almacenarse en cada router, ya que el campo TTL, y posiblemente también el campo de opciones, pueden cambiar.
Ethernet utiliza la técnica conocida como Comprobación de Redundancia Cíclica (CRC) (4 bytes). El

propósito del campo CRC es permitir que el adaptador de red del receptor, detecte los errores de bit de la trama. Los códigos CRC también se conocen con el nombre de códigos polinómicos, dado que se puede ver la cadena de bits que hay que enviar como si fuera un polinomio cuyos coeficientes son los valores 0 y 1 de la cadena de bits, interpretándose las operaciones realizadas con la cadena de bits según la aritmética de polinomios. Permite detectar determinados errores de bit en ráfaga. Si se detecta error, se descarta la trama.

b)
UDP proporciona una suma de comprobación aun cuando muchos protocolos de la capa de enlace (por ejemplo, Ethernet) también proporcionan mecanismos de comprobación de errores. La razón de ello es que no existe ninguna garantía de que todos los enlaces existentes entre el origen y el destino proporcionen un mecanismo de comprobación de errores (no es el caso de la letra de la pregunta); es decir, uno de los enlaces puede utilizar un protocolo de la capa de enlace que no proporcione comprobación de errores. Además, incluso si los segmentos se transfieren correctamente a través del enlace, es posible que se introduzcan errores de bit cuando un segmento se procesa y almacena en la memoria de un router. Por lo tanto, sí resulta necesario contar con dicha funcionalidad en el protocolo de capa de transporte.

Problemas Prácticos

Problema 1 (30 puntos)

Se desea implementar un servicio que retorne la IP pública desde donde se ha accedido a Internet. Para ésto se plantea generar un servidor que cuando se le realiza una consulta web, devuelve la IP desde donde se realizó la consulta. El servicio se accede a través de la URL `http://miIPpublica.com/getIP`.

Ejemplo de un intercambio para la solicitud, es el siguiente:
Solicitud desde el cliente:

```
GET /getIP HTTP/1.1
Host: miippublica.com
User-Agent: <identificador agente>
Accept: */*
```

Respuesta del servidor:

```
HTTP/1.1 200 OK
date: Fri, 07 Feb 2025 18:17:01 GMT
content-type: text/plain; charset=utf-8
Content-Length: 13
200.36.24.104
```

Se pide:
Implemente el cliente y el servidor en un lenguaje de alto nivel usando las primitivas de Sockets de la cartilla del curso. Dispone de funciones auxiliares para estructuras de datos, manipulación de strings, etc.

Solución
a)

Para la detección del final de la solicitud se utilizan los caracteres “\r\n\r\n”. Ésto permite al servidor determminar el final de la solicitud.

Los elementos que verifico para aceptar la solicitud es que el header Host sea ”miippublica.com” y que el get solicite objeto ”getIP”.

Para obtener la IP del cliente se utiliza la función de la cartilla `ip, port = client.getpeer().`

Servidor:

```
function server()
master = socket.tcp()
master.bind(*, 80)
server = master.listen()
while true do
    client, err = server.accept()
    thread.new(thread_getIP, client)
end
server.close() // inalcanzable, para cuando se de de baja el servidor
```

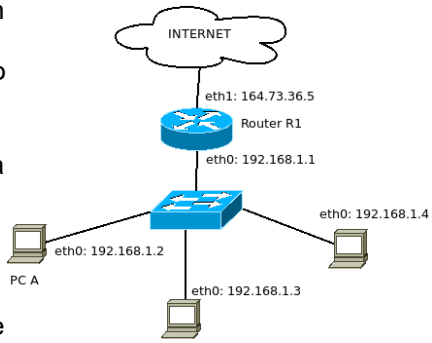
```
function thread_getIP(cli)
// leemos el get
request = ""
repeat
    fragment, err = cli.read()
    request += fragment
until request.sub(-4)=="\r\n\r\n" or err="closed"
if err=="closed" then
    cli.close()
    return
else
    host = request.match("Host: (.*)\r\n")
    get = request.match("GET /(.*) HTTP/1\r\n")
    date= date.now
    if host=”miippublica.com” and get=”getIP” then
        ip, port = client.getpeer()
        length = get_length(ip)
        response = “HTTP/1.1 200 OK \r\ndate: “.date.”\r\ncontent-type: text/plain;\r\
ncharset=utf-8\r\nContent-Length: “.length.”\r\n\r\n”.ip
        // enviamos la respuesta
        repeat
              response, err = cli.send(response)
        until response == "" or err=="closed"
        if err=="closed" then
              cli.close()
              return
        end
    end
    cli.close()
end
```

Cliente:

```
function cliente_getIP()
client = socket.tcp()
ip = gethost(miippublica.com)
client.connect(ip, 80)
getMsg = “GET /getIP HTTP/1.1\r\nHost: miippubllica.com\r\nUser-Agent: myApp\r\nAccept: */*\
r\n\r\n”
repeat
    response, err = client.send(getMsg)
until getMsg == "" or err=="closed"
if err=="closed" then
    client.close()
    return
end
response = ""
repeat
    fragment, err = cli.read()
    response += fragment
until response.sub(-4)=="\r\n\r\n" or err="closed"
if err==”closed”
    exit;
header = response.sub(1, response.pos(“\r\n”))
content_length = header.match(".*Content-lenth: (\d+)\r\n")
repeat
    fragment, err = cli.read()
    response += fragment
until response.length()-header.length()-2==content_length or err="closed"
if err==”closed”
    exit;
ip = response.sub(header-length()+2, response.length())
print(‘mi direccion publica es ‘.ip)
client.close()
```

Problema 2 (35 puntos)

Se cuenta con una red LAN privada que tiene acceso a Internet mediante un router que implementa NAT como se muestra en la figura.
Todas las PCs de la red LAN tiene configurada la `IP 192.168.1.1` como *default gateway* y como servidor DNS.



Considere el escenario donde la PC A realiza una solicitud HTTP para descargar la página web `www.redes.edu.uy/index.html`.

Se pide:

a) Enumere y describa todos los mensajes de capa 2, 3 y 4 que atraviesan la interfaz eth0 de la PC A. Especifique claramente direcciones, puertos y la función de cada mensaje.
Considere que no existe información de ningún tipo cacheada al ejecutar la solicitud.
Se sugiere utilizar una tabla con la siguiente información (cuando corresponda):
Tipo de mensaje – Cabezal de capa 2 – Cabezal de capa 3 – Payload (Detallado)

b) Muestre los mensajes (con direcciones y puertos) que atraviesan las interfaces eth0 y eth1 del router R1 cuando se trasmite la solicitud y respuesta HTTP.

c) Suponga que está capturando tráfico con Wireshark en R1:eth0 filtrando por protocolo ICMP cuando la PC A ejecuta ping -c 2 192.168.1.4. Detalle, si hay alguno, los paquetes que tendría su captura luego de la finalización del ping asumiendo que este es el único tráfico ICMP red y que todas las tablas de los dispositivos de red están vacías, pero no así las de los hosts.

Solución:

Parte a

Tipo de mensaje	Cabezal capa 2	Cabezal capa 3	Payload
ARP Request generado por la PC A	MAC orig: MAC_A_eth0 MAC dest: FF:FF:FF:FF:FF:FF	-	sender_mac: MAC_A sender_IP: 192.168.1.2 target_mac: 00:00:00:00:00:00 target_IP: 192.168.1.1
ARP Response generado por R1	MAC orig: MAC_R1_etho MAC dest: MAC_A_eth0	-	sender_mac: MAC_R1_eth0 sender_IP: 192.168.1.1 target_mac: MAC_A target_IP: 192.168.1.2
DNS Query	MAC orig: MAC_A_eth0 MAC dest: MAC_R1_etho	IP orig: 192.168.1.2 IP dest: 192.168.1.1	Datagrama UDP con: Puerto orig: 12345 Puerto dest: 53 en cuyo payload viaja la consulta DNS de tipo A por www.redes.edu.uy
DNS Reply	MAC orig: MAC_R1_etho MAC dest: MAC_A_eth0	IP orig: 192.168.1.1 IP dest: 192.168.1.2	Datagrama UDP con: Puerto orig: 53 Puerto dest: 1234 en cuyo payload viaja la

			respuesta DNS de la consulta anterior
Inicio de conexión TCP	MAC orig: MAC_A_eth0 MAC dest: MAC_R1_etho	IP orig: 192.168.1.2 IP dest: IP_fing	Segmento TCP con: Puerto orig: 12345 Puerto dest: 80 con la bandera SYN encendida.
Respuesta de conexión TCP	MAC orig: MAC_R1_etho MAC dest: MAC_A_eth0	IP orig: IP_fing IP dest: 192.168.1.2	Segmento TCP con: Puerto orig: 80 Puerto dest: 12345 con las banderas SYN y ACK encendidas.
Ack de conexión TCP	MAC orig: MAC_A_eth0 MAC dest: MAC_R1_etho	IP orig: 192.168.1.2 IP dest: IP_fing	Segmento TCP con: Puerto orig: 12345 Puerto dest: 80 con la bandera ACK encendida.
Pedido HTTP GET	MAC orig: MAC_A_eth0 MAC dest: MAC_R1_etho	IP orig: 192.168.1.2 IP dest: IP_fing	Segmento TCP con: Puerto orig: 12345 Puerto dest: 80 en cuyo payload viaja la solicitud HTTP: GET /index.html HTTP/1.1 (puede estar incluido en el segmento TCP ACK anterior)
Respuesta HTTP	MAC orig: MAC_R1_etho MAC dest: MAC_A_eth0	IP orig: IP_fing IP dest: 192.168.1.2	Segmento TCP con: Puerto orig: 80 Puerto dest: 12345 en cuyo payload viaja la respuesta HTTP: HTTP/1.1 200 OK y la página HTML
Reconocimiento del segmento TCP anterior	MAC orig: MAC_A_eth0 MAC dest: MAC_R1_etho	IP orig: 192.168.1.2 IP dest: IP_fing	Segmento TCP con: Puerto orig: 12345 Puerto dest: 80 con la bandera ACK encendida.
Fin de conexión TCP desde cliente	MAC orig: MAC_A_eth0 MAC dest: MAC_R1_etho	IP orig: 192.168.1.2 IP dest: IP_fing	Segmento TCP con: Puerto orig: 12345 Puerto dest: 80 con la bandera FIN encendida.
Respuesta de fin de conexión TCP	MAC orig: MAC_R1_etho MAC dest: MAC_A_eth0	IP orig: IP_fing IP dest: 192.168.1.2	Segmento TCP con: Puerto orig: 80 Puerto dest: 12345 con la bandera ACK encendida.
Fin de conexión TCP desde servidor	MAC orig: MAC_R1_etho	IP orig: IP_fing IP dest: 192.168.1.2	Segmento TCP con: Puerto orig: 80

	MAC dest: MAC_A_eth0		Puerto dest: 12345 con la bandera FIN encendida.
Respuesta de fin de conexión TCP	MAC orig: MAC_A_eth0 MAC dest: MAC_R1_etho	IP orig: 192.168.1.2 IP dest: IP_fing	Segmento TCP con: Puerto orig: 12345 Puerto dest: 80 con la bandera ACK encendida.

Parte b

Debido a que no es posible utilizar el espacio de direcciones privado que tiene asignada la LAN para comunicarse hacia Internet, el mecanismo NAT realiza una traducción de estas direcciones privadas en una dirección pública. Debido a que debe traducir varias direcciones privadas en una única dirección pública hacia afuera de la red LAN, debe utilizar una tabla de traducciones que incluye los número de puerto para realizar este mapeo.

En nuestro escenario, el router recibe por su interfaz eth0 un paquete IP con:
IP orig: 192.168.1.2
Puerto orig: 12345
IP dest: IP_redes
Puerto dest: 80

Al recibir el paquete, el router NAT genera un nuevo número de puerto origen (por ej. 5001) para el paquete, sustituye la dirección IP de origen por su dirección IP en la red pública: 164.73.36.5 y sustituye el número de puerto origen original (12345) por el nuevo número de puerto (5001).
Finalmente almacena el siguiente mapeo en su tabla de traducciones:

Red pública	Red privada
164.73.36.5, 5001	192.168.1.2, 12345

Por lo tanto, desde la interfaz eth1 se transmite a la red un paquete IP con los siguientes datos:
IP orig: 164.73.36.5
Puerto orig: 5001
IP dest: IP_redes
Puerto dest: 80

Cuando el router NAT recibe la respuesta del servidor web, recibe un paquete IP con los siguientes datos:
IP orig: IP_redes
Puerto orig: 80
IP dest: 164.73.36.5
Puerto dest: 5001

Luego, el router busca en su tabla de traducciones utilizando la dirección IP de destino y el puerto de destino y obtiene la dirección IP y puerto necesarios para hacer el cambio nuevamente.
Por lo tanto, el router NAT modifica esta vez la dirección IP y puerto de destino y se obtiene el siguiente paquete que será transmitido por la interfaz eth0 hacia la red privada:
IP orig: IP_redes
Puerto orig: 80

IP dest: 192.168.1.2
Puerto dest: 12345

Parte c

Debido a que las tablas de las PCs pueden tener información se asume que las tablas ARP están completas.
Como la tabla del switch está vacía, el primer ICMP echo request del ping va a ser reenviado por todas las interfaces del switch, llegando a R1:eth0. Luego, la respuesta a ese echo request solo será enviada por la boca conectada a PCA:eth0 ya que el mecanismo de self-learning del switch en el paso anterior guardó la asociación (MAC_PACA_eth0, boca-correspondiente). El siguiente y último echo request va a ser enviado solo por el link a la PC destino ya que en el previo echo reply el switch guardó la asociación (MAC_PACB_eth0, boca-correspondiente)

Por lo tanto al finalizar la captura se tendrá solo un paquete con el ICMP echo request de PC_A a PC_B.