Control 3
Redes
Plazo de entrega: 25 de junio 2025

José M. Piquer

Introducción

Las preguntas deben responderse en lenguaje directo, técnico y evitar las frases que suenan bien pero no dicen nada.

Un ejemplo sacado de chatGPT es: "Es importante asegurarse de que el enrutamiento, la configuración de NAT y las políticas de seguridad estén correctamente implementadas para garantizar una comunicación fluida y segura tanto dentro como fuera de la red corporativa".

Si se fijan, esa frase no dice nada concreto, nada que permita evaluar si Uds entienden la materia (sabemos que chatGPT NO la entiende).

Si Uds escriben algo así en el control, tendrán cero puntos por ello.

P1: TCP avanzado

1.1 Control de Flujo

El protocolo de Control de Flujo en TCP es una función importante del protocolo. Responda las siguientes preguntas:

- Un ingeniero argumenta que el control de flujo en TCP es necesario para que la ventana de recepción del protocolo no se sature más allá de su capacidad. Es decir, que cuando la ventana de recepción está llena, debe avisarle al enviador para que deje de enviar datos. Discuta esta afirmación y argumente: ¿está bien esa explicación?

- Otro ingeniero argumenta que incluso si la ventana de recepción está vacía, puede ser necesario detener al enviador para implementar bien el control de flujo. Discuta esta afirmación y explique.

- Cuando el control de flujo decide avisar que no hay espacio en el receptor, el enviador debe detener su flujo. ¿Qué ocurre entonces con la aplicación si sigue intentando enviar datos?
- Si el enviador recibe un aviso de control de flujo diciendo que el receptor no tiene más espacio de recepción, ¿qué debe hacer con la ventana de envío que todavía tiene pendiente? ¿debe seguir retransmitiendo esos paquetes? ¿o debe detener la retransmisión?

1.2 RTT y eficiencia de TCP

Responda las siguientes preguntas:

- En un TCP usando selective repeat, un ingeniero propone que podemos detectar que nuestro timeout es demasiado pequeño si empezamos a recibir ACKs duplicados, y entonces debemos agrandarlo. Otro ingeniero le argumenta que está equivocado, por que el receptor no enviará un ACK si el paquete ya fue recibido bien antes. ¿Quién tiene razón? Argumente.
- Un problema conocido de TCP es que se comporta mal en enlaces de alto RTT, debido a la regla de que sólo actualiza la ventana de congestión una vez por RTT. Explique por qué funciona así y discuta si se podría mejorar su tiempo de reacción en esos casos.
- Tenemos una conexión TCP sin congestión, pero mi computador tiene un enlace WiFi con alta interferencia, lo que le hace perder un 10 % de los paquetes. ¿Es posible que TCP no logre sacar el 90 % del ancho de banda? ¿Por qué?
- Si uno pudiera tener un computador con memoria infinita (y gratuita), ¿sería buena idea usar ventanas de envío y recepción infinitas con selective repeat? ¿Qué límites le pueden afectar?

P2: Redes IP

2.1 Encolamiento y Prioridades

Un tema difícil para IP es que trata a todos los paquetes iguales, y obviamente hay prioridades distintas cuando se trata de videoconferencias o de un mail. Otro problema es que el encolamiento en los routers introduce delays, pero si no tenemos memoria entonces generamos pérdidas. Responda las siguientes preguntas:

- Un ingeniero propone usar la prioridad del paquete para ordenar la cola: de esta forma siempre están en primera posición los paquetes con mejor prioridad. ¿Funcionaría esta propuesta? ¿Es buena o mala idea?
- ¿Cómo podríamos determinar el tamaño óptimo de la cola de espera de un router, como una forma de definir cuánta memoria ponerle?

2.2 IPv6

Conteste las siguientes preguntas:

- Una empresa lo contrata a Ud para conectarla a IPv6. ¿es mejor pedirle un prefijo al proveedor de Internet o conseguir uno propio para la empresa en LACNIC? ¿Cómo decidiría entre estas opciones?
- En IPv6 se elimina el checksum del header. ¿Qué pasa si tenemos un error de transmisión y se altera el paquete?
- IoT apareció mucho después de IPv6. ¿Qué cambios le haría a IPv6 si estuviera pensando en IoT?

2.3 Máscara de Red

Un equipo conectado a la red 192.168.10.0/23 con dirección 192.168.10.10, no logra conectarse a otro equipo conectado a la red 192.168.11.0/24 con dirección 192.168.11.11. Ambas redes están unidas por un sólo router que está bien configurado, con las IP 192.168.10.1 y 192.168.11.1.

Conteste las siguientes preguntas:

- Dibuje un diagrama de la red presentada y la tabla de rutas en el router
- Escriba la tabla de rutas que tiene 192.168.10.10
- Explique por qué la conexión no funciona
- Proponga una solución
- ¿El router debió detectar un error de configuración, o es una configuración válida?

P3: Ruteo

3.1 Tablas de Rutas

Anycast se implementa instalando un prefijo de red en varias redes físicas distintas, en diversas partes del mundo. Obviamente, Internet y el ruteo no fueron pensados para esta solución.

Responda las siguientes preguntas:

- Un router central de Internet puede entonces recibir anuncios de caminos totalmente distintos al mismo prefijo: uno en Brasil, otro en Chile, otro en África. ¿Por qué ese router es capaz de procesar bien estas alternativas y no necesita entender que es un anycast?
- ¿Cómo cree ese router que es el mapa de Internet donde está ese prefijo, sin considerar la posibilidad de anycast?
- Si estoy conectado con TCP a un host anycast y justo cambia la ruta elegida en ese momento y quedo conversando con otra réplica en otro lugar: ¿Qué pasa con mi conexión?

3.2 BGP-4

La idea de base de BGP-4 es intercambiar "vectores de caminos"llamados `AS PATH`. Explique cómo hace BGP-4 para no generar loops en las rutas, como le pasa a RIP que usa "vectores de distancias".

P4: MTU Path Discovery y UDP

En UDP, también sería útil descubrir el MTU Path, y así usar el paquete UDP más grande posible que no se fragmente. Existen algunas propuestas, y algunos sistemas operativos ya están tomando algunas decisiones de ese tipo, por ejemplo, MacOS suele enviar los paquetes UDP con la opción de no fragmentar.

Busque información en Internet y responda la siguientes preguntas:

1. La IA de Gemini me dijo: *Path MTU Discovery (PMTUD) for UDP involves a mechanism for UDP-based applications to dynamically determine the maximum transmission unit (MTU) size along a network path, preventing fragmentation. This is crucial because UDP is a connectionless protocol and doesn't handle reassembly of fragmented packets like TCP.*

   Discuta qué tan bien explicado está y argumente si encuentra algún error en su explicación.

2. Existe una propuesta de implementación para UDP, en que se le agrega la opción `IP_FINDPMTU` a la primitiva `getsockopt()`:

   ```
   mtu_path = sock.getsockopt(socket.IPPROTO_IP, socket.IP_FINDPMTU)
   ```

   En ese escenario, todos los paquetes UDP son enviados con el bit de no fragmentar, y si se recibe un paquete ICMP de error diciendo que el paquete es muy grande, el kernel recalcula el MTU PATH de ese socket, y puedo re-obtenerlo con `getosckopt()`. Cualquier intento de enviar con `sock.send()` un paquete más grande que el MTU PATH genera un error de *Message too big*.

   Modifique este código del cliente de eco UDP que usamos para la tarea, de modo que parta con el `PACK_SZ` que se le entrega de argumento, pero luego se adapte para usar siempre el MTU PATH que calcula el sistema si es más pequeño que el argumento.

   ```
   # En este otro thread leo desde stdin hacia socket:
   sz = 0
   while True:
       data = fin.read(PACK_SZ)
       if not data:
   ```

```
            break
        s.send(data)
        sz += len(data)

    s.send(b'')
    newthread.join()
    s.close()
```

3. ¿Es necesario modificar el código del thread receptor también? El thread receptor es:

```
def Rdr(s, fout):
    global PACK_SZ
    sz = 0
    s.settimeout(5)
    while True:
        try:
            data=s.recv(PACK_SZ)
        except:
            data = None
            print('Timeout')
        if not data:
            break
        fout.write(data)
        sz += len(data)
    fout.close()
```

4. Un problema clásico de MTU Path Discovery es que los paquetes ICMP tienden a ser bloqueados en Internet y nunca llegan al origen. Si este fuera el caso, en una conexión UDP es peor que para TCP, ya que no tenemos ACKs y cosas así que nos permitan detectar timeouts y concluir que el paquete se perdió. En el caso del ejemplo, si quisiéramos detectar que el tamaño de paquete que estamos usando, en realidad hace que se pierdan todos, ¿cómo modificaría el código para detectar esto y disminuir el tamaño para volver a probar?