C3 REDES

Nombre: Patricio Espinoza A. Sección: 1

P1 TCP Avanzado

1.1 Control de Fujo

- **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 esta llena, debe avisarle al enviador para que deje de enviar datos. Discuta esta afirmación y argumente: ¿está bien esa explicación?**
  R: La explicación está bien en términos generales ya que efectivamente cuando la ventana del receptor se satura, se debe avisar a la ventana del emisor para ajustar el flujo de envío de los paquetes. Esto es debido a que la ventana del receptor son buffers, y por tanto si están llenos los paquetes no podrán ser recibidos y deberán ser retransmitidos, lo que a su vez dependiendo del protocolo (Selective Repeat, Go-back, etc) puede generar aún más ancho de banda.

  Un punto a destacar es que no solo se avisa al emisor que se detenga cuando la ventana del receptor se llenó. Sino que también se puede ir informando/actualizando la información del tamaño de la ventana del receptor (si esta disminuyó o aumentó) tal que el emisor ajuste su ventana en consecuencia, por tanto es necesario para que la ventana del receptor no se sature, pero también para que la comunicación/sincronización de paquetes sea más eficiente que en una implementación sin control de flujo.

- **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.**
  R: Si podría llegar a ser necesario, este caso se da si es que el receptor no lee correctamente los datos recibidos y para evitar que sean perdidos, envía una señal al emisor para que detener el flujo de paquetes. El no leer correctamente los datos (receptor) se puede dar por una congestión en la pila de la red o lentitud en la aplicación, es decir, en ambos casos ocurre que el procesamiento por parte del receptor es más lento que la llegada de paquetes con datos, tan lento que se requiere pausar momentáneamente al emisor con tal de evitar que los paquetes enviados a posteriori sean perdidos y por consiguiente requieran ser retransmitidos.

- **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?**
  R: La aplicación debe bloquear la sección en donde se realizan la llamada send() (también se puede implementar tal que retorne un error) para evitar que en el emisor se escriban datos en el buffer de la ventana de envío, desbloqueándose únicamente cuando el receptor se lo indique. Aquí no hay paquetes perdidos que deban ser retransmitidos pues quedan sin enviarse hasta que la condición de ventana del receptor se actualice.

- **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?**
  R: La ventana de envío que tiene pendiente, es decir, aquellos paquetes con datos nuevos que no se han enviado, deben detenerse hasta que se reciba una señal. Para ello se puede enviar paquetes de prueba de forma periódica para comprobar si en el receptor se liberó espacio en la ventana y esta recibiendo paquetes (o si si sigue lleno). Solo en este caso se podrá renaudar la ventana de envío que estaba pendiente y continuar con el flujo.

  Por otra parte, tenemos que aquellas retransmisiones en curso también deberán ser detenidas pues la condición de detención aplica para todo lo que es enviado por el emisor. El protocolo TCP no reserva un espacio a estos paquetes retransmitidos en la ventana del receptor ya que el control de flujo es dinámico, y el tamaño de la ventana representa el espacio disponible en el momento (de una forma más simple, el paquete retransmitido rebotará al llegar donde el emisor pues no hay espacio en la ventana, así que es inútil enviarlo). En particular, en este protocolo el receptor descartará todo paquete recibido cuando la ventana es de 0, así que la retransmisión debe ser también debe ser detenida.

1.2 RTT y eficiencia de TCP

- **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, porque el receptor no enviará un ACK si el paquete ya fue recibido bien antes. ¿Quién tiene razón? Argumente.**
  R: El segundo ingeniero tiene razón, Selective Repeat solo enviará un ACK duplicado si hubo un paquete perdido o fuera de orden (aquí el protocolo lo almacena). Si el paquete fue bien recibido antes, entonces no tiene porque generar otro ACK (y por consiguiente duplicados).

  Es decir, los ACKs duplicados indican la pérdida de paquetes, más no que el timeout sea pequeño. Para determinar si es necesario incrementar el timeout porque este es muy corto, se deben estudiar las retransmisiones innecesarias seguidas de ACKs normales, en particular, fijándose en el RTT.

- **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.**
  R: El protocolo TCP actualiza la ventana de congestión una vez por RTT para evitar sobre cargar la red, esto también permite observar los cambios producidos por el impacto debido a la actualización. Una desventaja de esto es que al aumentar el RTT, los cambios tardan más en detectarse.

  Es posible mejorarlo con algoritmos como TCP BRR de Google o CUBIC los cuales ajustan el ritmo de envío basándose no solo en el RTT sino que también en estimaciones independientes mediante el ancho de banda disponible, solventando el problema anteriormente mencionado.

- **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?**
  R: Si es posible que ocurra esto debido a la alta interferencia y por consiguiente la perdida de paquetes, lo que lleva a que en la sincronización en la red entre el emisor y receptor mostrará que hubo pérdida de paquetes (pero no sabrán que es por la alta interferencia), por tanto, el protocolo tomará como que hay una congestión (cuando no hay realmente), disminuyendo el ancho de banda para evitar saturar la red.

- **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?**
  R: Dentro del enunciado se tienen ventanas infinitas las cuales permitirán enviar y recibir una cantidad de paquetes mayor a la normal, y a su vez memoria infinita, que permitirá tener estos buffers necesarios para las ventanas, como no habrá escasez de buffers tampoco ocurrirán perdidas y retransmisiones por desbordes en los buffers (esto se relaciona con aquellos paquetes que deben almacenarse en un buffer en el protocolo SR puesto que venían desordenados).

  Se deben considerar el ancho de banda disponible, los números de secuencia, y el procesamiento de la aplicación.

  El ancho de banda es un factor que no se menciona en el enunciado y por tanto es una limitante, mediante el ancho de banda y el delay se puede obtener el BDP, con el cual de por si ya se obtiene la ventana óptima, incrementarlas más allá de este punto no daría ningún beneficio a SR.

  Los números de secuencia no son infinitos pues están limitados, aunque sea una cantidad grande, en un contexto de ventanas infinitas se generará repetición de estos, lo que llevará a paquetes "repetidos" generando un desorden en la red, confusión y por tanto reenvió de paquetes que no correspondía. (Mal funcionamiento)
  El procesamiento de la aplicación también es importante, tal como se habló en la segunda pregunta de la sección 1.1, al tener ventanas de envío de paquetes en un contexto donde la aplicación tarda más de lo normal (ineficiente/lenta) en leer los datos y aceptar como tal un paquete, se enviará una señal de detener preventivamente la sincronización emisor/receptor. Considerando que ahora se tendrían ventanas infinitas, esto resultaría aún peor. La única opción de que esto llegará a ser más viable es que la memoria infinita del computador garantice un funcionamiento no solo normal sino que óptimo de la aplicación al leer los paquetes, pero esto probablemente no sea viable pues el tener tanta memoria generará un costo algorítmico mayor al momento de querer liberar los buffers de aquellos paquetes aceptados. Lo que lleva nuevamente al punto de que la aplicación sea lenta, incrementando la latencia.

  Todo esto permite concluir que no sería una buena idea usar ventanas de envío y recepción infinitas con SR.

P2

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?**
  R: La propuesta es básicamente aplicar colas de prioridad vistas en Sistemas Operativos. Esta efectivamente funcionaría y permitiría solventar el problema, sin embargo, también incluye desventajas tales como starvation, en donde si en un día se tienen muchas videoconferencias y a estas constantemente se les asigna una prioridad mayor a los correos entonces estos últimos nunca se transmitirán, lo cual tampoco sería ventajoso para una empresa con alto flujo. Una mejora sería determinar un ancho de banda dependiendo de la prioridad tal que se evite inanición.

- **¿Cómo podríamos determinar el tamaño óptimo de la cola de espera de un router, como una forma de definir cuanta memoria ponerle?**
  R: El tamaño óptimo se puede tener mediante el BDP, el cual es el producto entre el ancho de banda y el RTT, y que representa la cantidad de datos que pueden estar activos en el flujo de la red.

  Una cola más grande que el BDP aumentará el delay ya que habrán muchos paquetes en el buffer. Una cola más pequeña que el BDP aumentará las pérdidas.

  Respecto a la memoria, esta debe ser como mínimo lo suficiente como para poder almacenar una cola capaz de mantener el flujo de los datos en la red, es decir, la memoria debe ser capaz de brindar espacio suficiente para los buffers de la cola que permite almacenar los datos enviados considerando el BDP calculado según los datos de la red.

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?**
  R: Para decidir entre ambas opciones tomaría en cuenta la independencia requerida por la empresa, el tamaño y escalabilidad requerida de acuerdo al tipo de empresa, y costos o complejidad deseados.

  Una ventaja de pedir un prefijo al proveedor de internet es que resulta más sencillo y barato en costos de red, sin embargo, se genera una dependencia pues en caso de tener que cambiar de proveedor, la red deberá ser reconfigurada. Por otra parte, conseguir uno propio en LACNIC dará independencia y direcciones globales estables, pero requiere cumplir con ser un sistema autónomo, lo cual si es una empresa pequeña puede no ser necesario, y por ende una opción más sencilla sería simplemente solicitarlo con el proveedor.

  Preliminarmente en este caso, consultaría primero con la empresa los proveedores que tienen, y su planificación a futuro. También podría ser interesante considerar si la ubicación geográfica es en una zona urbana, en desarrollo, o rural para evaluar esta dependencia ya que en este último caso un proveedor puede presentar fallas frecuentes y por tanto sería mejor opción LACNIC.

  Otro punto también es la complejidad de la red y si su infraestructura es vital, en cuyo caso requiere una estabilidad garantizada (LACNIC). En caso contrario, donde la red no es tan compleja ni requerirá una gran escalabilidad se puede optar por el prefijo del proveedor sin problemas.

- **En IPv6 se elimina el checksum del header. ¿Qué pasa si tenemos un error de transmisión y se altera el paquete?**
  R: Si se altera el paquete no habría problema pues IPv6 delega la tarea de estos errores a capas superiores como TCP/UDP que tienen su propio checksum. Asimismo, el enlace (Ethernet o Wifi) también tiene su propio checksum que al detectar el error descartará el paquete.

- **IoT apareció mucho después de IPv6. ¿Qué cambios le haría a IPv6 si estuviera pensando en IoT?**
  R: El IoT tiene necesidades más específicas, tales como la simplicidad y bajo consumo. Para ello, sería útil que IPv6 tuviera un header más compacto ya sea reduciendo los campos que utiliza o con un formato más comprimido con tal de disminuir la latencia y congestión (consumo). Además, los prefijos de las direcciones podrían ser más cortos, lo que en una red de IoT con múltiples sensores permite un ruteo mucho más escalable (simplicidad). Otro punto fuertemente ligado a IoT es el multicast eficiente, en donde se debe enviar un paquete desde un emisor a varios receptores distintos sin que hayan múltiples copias en cada sensor (simplicidad y consumo).

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**

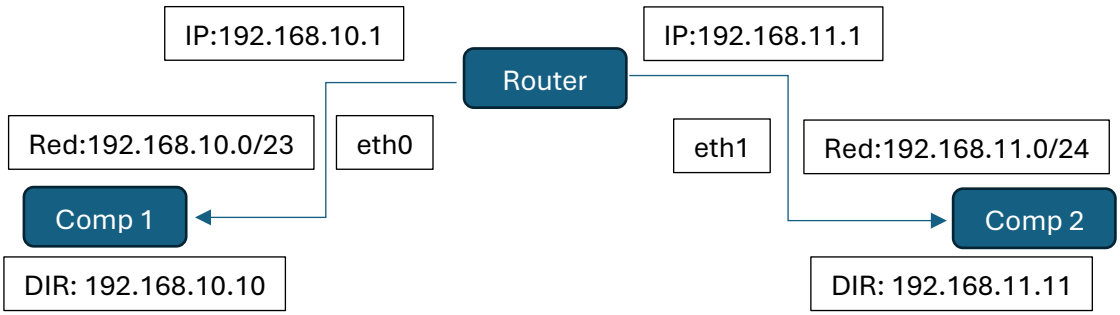

Tabla de Rutas		
Prefijo	**GW**	**IF**
192.168.10.0/23	**DIR**	**eth0**
192.168.11.0/24	**DIR**	**eth1**
0.0.0.0./0	**NA**	**NA**

Si “bien configurado” implica una conexión a internet con su respectiva interfaz, entonces sería similar al ejemplo de la actividad 7 del EOL pero no se especifica esta información.

0.0.0.0./0	**200.1.1.1**	**eth0**

- **Escriba la tabla de rutas que tiene 192.168.10.10**

Tabla de Rutas para 192.168.10.10		
Prefijo	**GW**	**IF**
192.168.10.0/23	**DIR**	**eth0**
0.0.0.0/0	**192.168.10.1**	**eth0**

- **Explique por qué la conexión no funciona**
  R: No se puede conectar al otro equipo (Comp 2) porque esta en el host 192.168.11.11 con una mask /24, es decir, en otra red. El dispositivo que desea conectarse (Comp 1) con host 192.168.10.10 está en otra red. La conexión que tiene esta red es con el router más no con el otro dispositivo, y por tanto generando esta falla.

- **Proponga una solución**
  R: La solución es arreglar el enrutamiento para que sea coherente, una opción es cambiarla la máscara del dispositivo al que deseamos conectarnos a /23 para que coincida con la del dispositivo que quiere conectarse. La otra es hacer esto mismo al revés (ambas con /24), el objetivo es dejar ambas con la misma máscara.

- **¿El router debió detectar un error de configuración, o es una configuración válida?**
  R: Lo enunciado es una configuración válida, cada máscara se configuro localmente para cada dispositivo correctamente y esto no representa un problema de invalidez.

  En cuanto a la detección del error, esto no es posible para el router ya que este solo confirma que ambas redes son válidas y actúa como intermediario enrutando según lo indicado. El router no realiza acciones de resguardo si hay una asimetría entre configuraciones de máscaras.

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?**
  R: Es capaz de procesar bien las alternativas ya que su función es únicamente enrutar eficientemente, por tanto, solo reconoce las múltiples rutas que le llegan hacia un mismo prefijo y actúa como intermediario. El router no comprenderá o entenderá que se está realizando un anycast ya que no es relevante para él, al momento de evaluar que ruta es más óptima para el enrutamiento solo se fijará en aquella que represente un menor costo.

- **¿Cómo cree ese router que es el mapa de Internet donde está ese prefijo, sin considerar la posibilidad de anycast?**
  R: El router creerá que el prefijo anycast pertenecerá a una sola red, es decir, un solo destino con múltiples rutas disponibles para llegar a este. En cada caso se encargará de que los paquetes vayan por el mejor camino pero no considerará la posibilidad de que hayan varias replicas físicas de la red apuntando al mismo prefijo.

- **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?**

  R: Si se cambia la ruta entonces la conexión se rompe ya que TCP depende tanto de la IP (emisor-receptor) como de los puertos para el flujo de envío de paquetes, la razón de que la nueva replica no sirva es que pierde el estado en el que encontraba la conexión previamente.

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".

R: BGP-4 evita los loops de ruteo usando el campo AS PATH, en donde se listan todos los sistemas autónomos que un anuncio de ruta ha atravesado. Si un router detecta su AS en el AS_PATH, descarta la ruta evitando el loop.

RIP por otro lado solo se fija en la cantidad de saltos (hop count), sin saber por donde pasó la ruta anteriormente y dando la posibilidad de entrar en un Loop sin detectarlo.

P4 MTU Path Discovery y UDP

1. La afirmación está bien explicada pero contiene un error ya que si bien PMTUD evita fragmentación, el decir que UDP no realizar el reensamblaje como TCP no es del todo correcto. Tanto IPv4 como IPv6 reensamblan los fragmentos a nivel de la IP independientemente del protocolo (TCP o UDP), los protocolos no reensamblan.

2. Lo que se desea hacer es iniciar con un tamaño máximo, disminuirlo automáticamente si el kernel detecta que supera el MTU real y volver a enviar el fragmento con un nuevo tamaño seguro.

```
PMTU = s.getsockopt(socket.IPPROTO_IP, socket.IP_FINDPMTU)
# PACK_SZ como mínimo entre MTU y argumento
PACK_SZ = min(PACK_SZ, PMTU)

sz = 0

while True:
  data = fin.read(PACK_SZ)
  if not data:
    break
  try:
    s.send(data)
    sz += len(data)
  except socket.error as e:
    if e.errno == socket.EMSGSIZE:  # Error "Message Too Big"
      # Actualizar PMTU y reducir PACK_SZ
      PMTU = s.getsockopt(socket.IPPROTO_IP, socket.IP_FINDPMTU)
      PACK_SZ = min(PACK_SZ, PMTU)
      continue
    raise  # Otro error
```

3. No es necesario ya que el receptor solo utiliza la variable global PACK_SZ, la que se va a ir actualizando de forma que no haya fragmentación al realizar el envío con send(PACK_SZ) en el emisor. Por tanto el receptor al hacer recv(PACK_SZ) no encontrará fragmentaciones.

4. Como al volver a probar se pierden datos se puede definir una cantidad de veces a probar, manteniendo que cada vez que se prueba se reduce PACK_SZ. En particular, en el receptor podemos crear un ACK personalizado de 1 byte para confirmar entregas.

```
TIMEOUT = 2.0  # Segundos sin respuesta para asumir pérdida
MAX_ATTEMPTS = 3
attempts = 0
while True:
  data = fin.read(PACK_SZ)
  if not data:
    break
  try:
    s.send(data)
    s.settimeout(TIMEOUT)
    ack = s.recv(1)  # Esperar confirmación
    sz += len(data)
    attempts = 0  # Resetear contador si hay respuesta
  except socket.timeout:
    attempts += 1
    if attempts >= MAX_ATTEMPTS:
      # Asumir que ICMP está bloqueado y reducir PACK_SZ
      PACK_SZ = max(PACK_SZ // 2, 512)  # Independiente del kernel y del ICMP
      attempts = 0
  except socket.error as e:
    if e.errno == socket.EMSGSIZE:
      PMTU = s.getsockopt(socket.IPPROTO_IP, socket.IP_FINDPMTU)
      PACK_SZ = min(PACK_SZ, PMTU)
    raise
```