Redes de Computadoras

Primer Parcial – 28 de setiembre de 2018

(ref: solprc20180928.odt)

Instrucciones

- **Indique su nombre completo y número de cédula en cada hoja.**
- **Numere todas las hojas e indique en la primera la cantidad total de hojas que entrega.**
- **Escriba las hojas de un solo lado y utilice una caligrafía claramente legible.**
- **Comience cada pregunta en una hoja nueva.**
- **Sólo se responderán dudas de letra. No se responderán dudas de ningún tipo durante los últimos 30 minutos de la prueba.**
- **La prueba es individual y sin material. Apague su teléfono celular mientras esté en el salón de la prueba.**
- **Duración: 2 horas. Culminadas las 2 horas, el alumno no podrá modificar de ninguna forma las hojas.**
- **Justifique todas sus respuestas.**

Pregunta 1 (3 puntos)

a) Describa la causa por las que se produce congestión en la red.
b) Nombre y explique dos síntomas que podemos ver en una red cuando está congestionada.
c) Describa brevemente que mecanismo utiliza TCP para detectar estos síntomas.

Solución

a) La congestión se produce cuando un nodo o link debe transportar mas información de la que soporta. Por ejemplo, una causa de congestión se da cuando un router recibe datos a una tasa mayor a la que puede enviarlos por lo que debe almacenar los paquetes en sus buffer. Si esta situación se mantiene estos buffers crecerán consistentemente.
b) Hay dos síntomas principales:
   1. altos valores de retardo debido a que los paquetes deben esperar por largo tiempo en las colas de los nodos (retardo de cola).
   2. pérdidas de paquetes debido a que las colas son de tamaño finito y al llenarse deben descartar paquetes.
c) Debido a que en TCP/IP no se cuenta con soporte explícito de la red para detectar congestión, los hosts finales deben inferir un estado de congestión observando localmente el comportamiento de la red. TCP en particular se basa en la pérdida de paquetes para inferir que la red se encuentra congestionada. Los eventos de pérdida en TCP son el vencimiento del temporizador y/o la recepción de 3 ACKs duplicados.

Pregunta 2 (3 puntos)

Dispone de un enlace corto de 10 metros a través del cual un emisor puede transmitir a una velocidad de 150 bits/segundo en ambos sentidos. Suponga que los paquetes de datos tienen una longitud de 100.000 bits y los paquetes que contienen solo comandos de control (por ejemplo, ACK o de acuerdo) tienen una longitud de 200 bits. Suponga también que hay N conexiones en paralelo y que cada una utiliza 1/N del ancho de banda del enlace. Considere ahora el protocolo HTTP y suponga que cada objeto descargado es de 100 kbytes de largo y que el objeto inicialmente descargado contiene 10 objetos referenciados procedentes del mismo emisor. ¿Tiene sentido en este caso realizar descargas en paralelo mediante instancias paralelas de HTTP no persistente? Considere ahora HTTP persistente. ¿Cabe esperar alguna ventaja significativa respecto del caso no persistente? Justifique y explique su respuesta.

Solución

Primero consideremos las descargas paralelas vía conexiones no persistentes. La descarga paralela nos habilitará 10 conexiones no persistentes compartiendo los 150 bits/seg de ancho de banda. Así, cada una obtendrá 15 bits/seg.

Notar que cada objeto es de 100 KBytes = 800.000 bits, por lo que serán necesarios 8 paquetes de datos.

Esto nos da un tiempo total dado por (Tp es el tiempo de propagación en el enlace):

- el establecimiento de la conexión TCP (three-way handshake) que incluye el pedido (GET) inicial: 200/150+Tp + 200/150+Tp + 200/150+Tp segundos

- la descarga del objeto inicial, dividido en 8 paquetes mas 8 ACKs: (100.000/150+Tp + 200/150+Tp)*8 = (100.000*8)/150 + (200*8)/150 + Tp*16

- los 10 inicios de conexión TCP en paralelo: 200/(150/10)+Tp + 200/(150/10)+Tp + 200/(150/10)+Tp

- las 10 descargas en paralelo: (100.000/(150/10)+Tp + 200/(150/10)+Tp)*8 = (100.000*8)/(150/10) + (200*8)/(150/10) + Tp*16

- total = **58828** + 38*Tp segundos

Ahora considerando una conexión HTTP persistente el tiempo total requerido es:

- el establecimiento de la conexión TCP (three-way handshake) que incluye el pedido (GET) inicial: 200/150+Tp + 200/150+Tp + 200/150+Tp

- la descarga del objeto inicial, dividido en 8 paquetes mas 8 ACKs: (100.000/150+Tp + 200/150+Tp)*8 = (100.000*8)/150 + (200*8)/150 + Tp*16

- Las 10 descargas consecutivas: 10*((100.000/150+Tp + 200/150+Tp)*8) = (100.000*8)/15 + (200*8)/15 + 160Tp

- Los pedidos GET se incluyen en los segmentos ACK del envío anterior

- total =**58788** + 179*Tp segundos.

Asumiendo que la velocidad de la luz es 300*10^6 m/seg, entonces Tp=10/(300*10^6)=0.03 microsegundos. Por lo que Tp es despreciable respecto al retardo de transmisión.
Entonces podemos ver que usar conexiones HTTP persistentes no tiene una significante ventaja (menos del 0,1%) respecto a las no persistentes con conexiones paralelas.

Pregunta 3 (4 puntos)

a) Defina los sockets, y cuál es su utilidad.

b) ¿Qué protocolos soporta? Describa someramente las diferencias.

c) Considere una aplicación servidor. Para cada protocolo considerado en la respuesta anterior, describa las primitivas que se deben ejecutar para atender solicitudes de clientes.

Solución

a) Los sockets son una interfaz que permite la comunicación entre procesos que pueden estar corriendo tanto en el mismo como en diferentes sistemas finales (*hosts*).

b) Soporta protocolos de capa de transporte, TCP y UDP. Los sockets que están asociados al protocolo TCP son orientados a conexión y garantizan la transmisión sin errores ni omisiones y además, llegará a destino en orden. Los sockets que están asociados al protocolo UDP no son orientados a conexión, por lo que puede que no lleguen todos los mensajes e inclusive, puede que no lleguen en el orden correcto.

c) TCP:
- master = socket.tcp() → se crea el socket
- master.bind(address, port) → se establece la dirección local y se asigna un puerto.
- server = master.listen() → convierte un socket master en socket server, capaz de aceptar conexiones
- client = server.accept() → Espera a que se establezca una conexión.
- data = client.receive(timeout) → Realiza una lectura en un socket conectado. Devuelve la información disponible en el stream en data.
- close() → cierra el socket y libera recursos asociados.

UDP:
- skt = socket.udp() → se crea el socket
- skt.bind(address, port) → se establece la dirección local y se asigna un puerto.
- skt.receive(timeout) → realiza la recepción de un datagrama. El parámetro timeout indica el tiempo en segundos disponible para realizar una lectura.
- close() → cierra el socket y libera recursos asociados.

Problema 1 (10 puntos)

Considere esta topología en cadena, donde los enlaces son full-dúplex, y cada nodo utiliza la técnica de almacenamiento y reenvío:

A ---- B ---- C ---- D ---- E

A envía paquetes a E usando un RDT. Cada enlace puede trasmitir un paquete por segundo, y se pueden despreciar todas las fuentes de retardo excepto el de trasmisión.

a) ¿Cuál es el RTT entre A y E?

b) ¿Cuál es el throughput de un protocolo stop-and-wait que se ejecuta en A en ausencia de pérdidas?

c) Si A decide utilizar un protocolo de ventana deslizante, ¿cuál es el tamaño óptimo de la ventana? ¿Cuál es el throughput que se consigue con ese tamaño de ventana?

d) Suponga que A utiliza el protocolo de ventana deslizante con un tamaño de ventana cuatro. Manteniendo la hipótesis de que no existen pérdidas, ¿cuál es el throughput? ¿Cuál es la utilización del enlace B-C? (por ejemplo, si el enlace puede trasmitir 100 paquetes por segundo, y el throughput es 10 paquetes por segundo, la utilización es 0.1).

e) Considere ahora un protocolo de ventana deslizante con el tamaño de ventana hallado en la parte c). Suponga que debido a un virus se descartan los paquetes con número de secuencia impar; el protocolo de ventana deslizante arranca en 1. Asuma además que el emisor tiene un timeout de 40 segundos, y que el receptor envía a un buffer los paquetes fuera de orden hasta poder entregarlos en orden a la aplicación.

   ¿Cuántos paquetes tiene este buffer 35 segundos después que el emisor comenzó a trasmitir el primer paquete?

Solución

a) Se debe contar el retardo generado por la transmisión en cada enlace del paquete de datos mas el retardo del paquete de *acknowledge.*
Por lo tanto el RTT entre A y E es de 8 segundos.

b) El protocolo stop-and-wait consiste en que no se envía un nuevo paquete hasta que no se recibe la confirmación de recepción del paquete anterior. Como el RTT es de 8 segundos, entre el envío de 2 paquetes consecutivos se deberá esperar 8 segundos. Por lo tanto, la tasa de envío (*throughput*) *es de* 1/8 paquetes por segundo.

c) La ventana óptima será aquella que me permita maximizar la cantidad de paquetes siendo enviados (paquetes en viaje). Dado el RTT calculado en la parte a) y dado que el tiempo de transmisión es de 1 segundo, el tamaño óptimo de la ventana es de 8 paquetes. Con esta ventana se obitene un tasa de envío de 1 paquete por segundo.

d) Con un tamaño de ventana de 4 paquetes, se podrán enviar los 4 paquetes de la ventana en 4 segundos, por lo que para la confirmación del primer paquete enviado (y poder comenzar a enviar el primer paquete de la nueva ventana) se deberá esperar 4 segundos. Por lo que se envian 4 paquetes cada 8 segundos. Esto nos da una tasa de envío de 0,5 paquetes por segundo.

Como el enlace B-C puede transmitir 1 paquete por segundo, la tasa de utilización será de 0,5.

e)
Si consideramos la ventana deslizante con el funcionamiento descrito en el curso para *Selective Repeat* tendremos:
Con un tamaño de ventana de 8, el emisor envía los paquetes con número de secuencia 1 a 8 en los primeros 8 segundos. Pero solo obtiene 4 ACKs, los correspondientes a los paquetes 2, 4, 6 y 8. Como no obtiene el ACK del paquete 1, no puede avanzar la ventana. El emisor deja de transmisitr hasta que se vence el timer del primer paquete en el segundo 40.
Por lo tanto, en el segundo 35 el emisor ha transmitido 8 paquetes de los cuales solo 4 han sido recibidos por el receptor. Pero como estos 4 paquetes están en desorden, ninguno a sido entregado y el buffer del receptor tiene 4 paquetes.

Si consideramos una ventana donde permitimos ACKs fuera de orden para avanzar tendremos:
Con un tamaño de ventana de 8, el emisor envía los paquetes con número de secuencia 1 a 8 en los primeros 8 segundos. Pero solo obtiene 4 ACKs, debido al descarte de los paquetes impares 1, 3, 5, y 7. Por lo tanto, el emisor transmite 4 paquetes mas (9 a 12) en los siguientes 8 segundos , recibiendo solo 2 ACKs. En los siguientes 8 segundos solo transmite 2 paquetes (13 y 14) y recibe 1 solo ACK. En los siguientes 8 segundos transmite solo 1 paquete con número de secuencia 15.
Hasta el momento han transcurrido 32 segundos y el emisor no recibe mas ACKs ya que el paquete 15 fue descartado. El emisor deja de transmisitr hasta que se vence el timer del primer paquete en el segundo 40.
Por lo tanto, en el segundo 35 el emisor ha transmitido 15 paquetes de los cuales solo 7 han sido recibidos por el receptor. Pero como estos 7 paquetes están en desorden, ninguno a sido entregado y el buffer del receptor tiene 7 paquetes.