Redes de Computadoras

Solución – 3 de agosto de 2013

(ref: scyoredes1307.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.**
- **Sólo se contestarán dudas de letra. No se aceptarán dudas de ningún tipo los últimos 30 minutos del examen.**
- **El examen es individual y sin material. Apague su celular mientras este en el salón del examen.**
- **Es obligatorio responder correctamente al menos 15 puntos en las preguntas teóricas y 20 de los problemas prácticos.**
- **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 (8 puntos)

a) Describa el algoritmo conocido como TCP Slow Start y su vinculación con el control de congestión.
b) Nombre y de una breve descripción de otros dos algoritmos que cooperen en el control de congestión TCP (no más de dos oraciones por cada uno).

Solución:

a) Al comienzo de la conexión, TCP tiene una tasa de transferencia baja, que se incrementa en forma exponencial hasta que se alcanza un valor de umbral *ssthesh o* hasta que se registre el primer evento de pérdida. Inicialmente, la ventana de congestión tiene un valor *cwnd* = 1 MSS, y se va duplicando cada vez que pasa el tiempo de un RTT, implementado de forma tal que cada vez que se recibe un ACK se incrementa en 1 MSS el valor de *cwnd*.

   El control realizado por la ventana *cwnd* está fijado por la siguiente limitante: *LastByteSent-LastByteAcked* < *cwnd*, por lo que si aumenta *cwnd*, tendrá una ventana mayor para el envío de datos.

   Resumiendo, inicialmente tiene una taza de transferencia baja, pero se incrementa rápido en forma exponencial.

b) *Congestion Avoidance*, mientras no se reciban ACKs duplicados, la ventana de congestión *cwnd* se incrementa linealmente en una unidad por RTT. Esto se debe a que la recepción de ACK duplicados, en general se debe a pérdida de paquetes.
   Fast Recovery, TCP retransmite el segmento que ha recibido tres ACKs duplicados y espera la llegada del ACK correspondiente, antes de volver a *Congestion Avoidance*. En caso de no recibir el ACK, entonces pasa a *Slow Start*.

Pregunta 2 (6 puntos)

La ventana de congestión (cwnd) típica al inicio de una conexión TCP es de 4KB. Describa el impacto que tendría cuadruplicar su tamaño en una LAN y en Internet. Recomiende o desaconseje la aplicación de este cambio a los *stacks* TCP/IP de los sistemas operativos.

Solución:

En Internet la mayoría de las transacciones Web son de corta duración y por lo tanto terminan antes de finalizar el slow start. Esto genera que la ventana de congestión inicial sea un parámetro crítico en la velocidad de los flujos Web. En particular reduciría la latencia de las transacciones que se completan en el estado slow start, también permitiría aumentar el tamaño de las páginas Web sin impactar en el tiempo de carga de la página. Tiene menos relevancia en una LAN donde la congestión no suele ser un problema importante. Esta modificación forma parte de un conjunto de cambios que buscan mejorar el throughput de TCP, y que están siendo incorporadas en distintos sistemas operativos, por ejemplo en el kernel de Linux 3.2.

Pregunta 3 (6 puntos)

a) En redes TCP/IP, ¿cuál es la interfaz utilizada por una aplicación, que le permite hacer uso de los servicios de la capa de transporte?
b) ¿Qué atributos mínimos son necesarios para identificar una comunicación TCP?
c) El protocolo UDP no es confiable ni orientado a conexión, ¿por qué usaría usted UDP en lugar de usar directamente IP para comunicarse con otro *host*?

Solución:

a) La interfaz utilizada es una API denominada sockets. Esta interfaz es brindada por el sistema operativo, y permite el intercambio de la información de capa de red con las aplicaciones.
b) Los atributos requeridos para definir un socket TCP: dirección IP local, dirección IP remota, número de puerto local y número de puerto remoto, que permiten al sistema operativo identificar los procesos que intercambian datos.
c) UDP provee una interfaz de aplicación para IP. Como se indica no soporta confiabilidad, control de flujo o recuperación de errores para IP. Lo que permite es la posibilidad de "multiplexar/demultiplexar" para enviar y recibir datagramas, usando puertos para dirigir los datagramas a las aplicaciones correspondientes. De esta forma se generaliza la utilización de protocolos de transporte por parte de las aplicaciones. Si se utilizara IP directamente, cada aplicación debería implementar los mecanismos de mux/demux mencionados anteriormente.

Pregunta 4 (10 puntos)

a) Explique los siguientes conceptos para la capa de enlace: *Framing*, *Link Access*, *Half-duplex* y *Full-duplex*.
b) El uso de medios de acceso compartido genera colisiones en la transmisión. Mencione un protocolo para cada una de las siguientes categorías de protocolos:
   - Particionamiento del canal (*Channel Partitioning*).
   - Acceso aleatorio al canal (*Random Access*).
   - Toma de turnos (*Taking Turns*).

Solución:

a) *Framing* se refiere a encapsular el datagrama de capa de red en un frame de capa de enlace, agregando un cabezal (header) y trailer.
   Links Access involucra el acceso al medio en caso de medios compartidos.
   Half-duplex indica que ambos extremos de un enlace punto a punto pueden transmitir pero no al mismo tiempo.
   Full-duplex indica que ambos extremos de un enlace punto a punto pueden transmitir al mismo tiempo.
b) Particionado del canal (*Channel Partitioning*): TDM, FDM, CDMA.
   Acceso aleatorio al canal (*Random Access*): ALOHA, CSMA/CD.
   Toma de turnos (*Taking Turns*): token ring.

Pregunta 5 (10 puntos)

Defina y explique brevemente los siguientes conceptos y describa como interactúan entre si: **protocolo de enrutamiento**, **tabla de reenvío o *forwarding***, **protocolo IP** y **protocolo ICMP**.

Solución:

El **protocolo de enrutamiento** permite a los routers intercambiar información topológica, y de esta forma, utilizando un algoritmo específico (link-state o distance vector), construir rutas hacia diferentes destinos en la red.

La **tabla de reenvío o forwarding** es generada a partir de los protocolos de enrutamiento, y establece que puerto de salida se debe utilizar para cada destino o rango de destinos de un datagrama.

El **protocolo IP** (Internet Protocol) define reglas para la transmisión de datagramas, a través de distintas redes físicas. El protocolo define, entre otros elementos:
- un sistema de direccionamiento, permitiendo asignar un identificador a cada interfaz de red,
- formato de los datagramas utilizados
- convenciones para el manejo de los datagramas.

ICMP (Internet Control Message Protocol) es un protocolo de control para IP; el mismo permite la notificación de errores y señalización de eventos ocurridos. Por ejemplo, en caso de que un datagrama tenga un valor de TTL =

0, se notificará al origen. ICMP es considerado un protocolo de capa de red, pero sin embargo se transporta como carga útil en IP.

Problemas Prácticos

Problema 1 (30 puntos)

Considere una red P2P de almacenamiento distribuido de archivos donde cada nodo conoce las direcciones IP de un subconjunto de los nodos que participan en la red (sus "vecinos").

a) Implemente, en un lenguaje de alto nivel y utilizando conexiones TCP, la o las funciones necesarias para realizar la búsqueda por coincidencia exacta de nombre de archivo entre todos los nodos que participan en la red P2P, con la menor cantidad de mensajes posible. La búsqueda debe devolver la dirección IP del nodo que almacena el archivo buscado.

b) Implemente la búsqueda anterior suponiendo que:

- Los archivos se encuentran ordenados alfabéticamente según la IP del nodo,es decir, que si un archivo A tiene un nombre alfabéticamente anterior a un archivo B, entonces esta almacenado en un nodo con una dirección IP igual o inferior a la del nodo que almacena al archivo B.
- Cada nodo solo conoce a los nodos de la red P2P con las IPs inmediatamente superior e inmediatamente inferior a la propia.

Se pueden utilizar las siguientes primitivas (y otras que usted considere necesarias):

1. `ObtenerVecinos()` que devuelve un conjunto de direcciones IP de los nodos la red P2P a los que conoce el nodo que la ejecuta.
2. `ObtenerSiguienteNodo(DirecciónIP dip)` que dada la dirección IP de un nodo participante de la red P2P, devuelve la dirección IP del vecino más cercano a ese nodo.

Problema 2 (30 puntos)

Se consideran dos nodos A y B directamente conectados.

a) Se envía un paquete ICMP echo request de tamaño P con el campo TTL>1 desde A a B.
   i. Describa el comportamiento del nodo B cuando reciba el paquete.
   ii. En base a esta respuesta, formule una expresión para el Round Trip Time entre A y B, considerando todos los componentes del retardo.
   iii. Suponga ahora que los nodos tienen infinitos recursos, y reformule la expresión del Round Trip Time. Analice los componentes constantes y variables en dicha expresión, y dibuje un diagrama de la función resultante.

b) Suponga ahora A envía a B una secuencia muchos paquetes ICMP echo request de tamaño variable Pi.
   i. Describa un mecanismo para medir el ancho de banda de un enlace utilizando este método.
   ii. Una herramienta existente para medir ancho de banda utilizando este mecanismo envía muchos paquetes de cada tamaño Pi, ¿por qué?

Solución Problema 1

Problema 1
a) Usamos 2 mensajes: “solicitud” (contiene un nombre de archivo y la ip del emisor original) y “respuesta” (contiene la IP donde se encuentra el archivo)

```
// Esta es la función invocada para acceder al servicio.
find (file) {
      if mi_archivo(file) return mi_ip
      for each ip in ObtenerVecinos() {
            skt = connect(ip)
            skt.send(construir_solicitud(mi_ip, file))
            respuesta=skt.read()
            skt.close()
            ip_en_respuesta = contiene_ip(respuesta)
            if (ip_en_respuesta) {
                  return ip_en_respuesta
            }
      }
}

//procesa mensajes que implementan el servicio
atender_solicitud(client) {
      solicitud = client:receive()
      file=parsear_solicitud_file(solicitud)
      ip_origen = parsear_solicitud_ip(solicitud)
      //ignorar si no viene por el camino mas corto (reverse path forwarding)
      if ( client != ObtenerSiguienteNodo(ip_origen)) {
            return
      }
      if mi_archivo(file) {   //disponible localmente
            client.send(contruir_respuesta(mi_ip()))
      } else { // salir a buscar
            for each ip in ObtenerVecinos() {
                  if ip != client.ip {  // no transmitir hacia atras
                        skt = connect(ip)
                        skt.send(construir_solicitud(file))
                        respuesta=skt.read() //puede devolver vacio si cierran
                        skt.close()
                        ip_en_respuesta = contiene_ip(respuesta)
                        if (ip_en_respuesta) {
                              client.send(contruir_respuesta(ip_en_respuesta))
                              return
                        }
                  }
            }
      }
}

//servidor de sockets para recibir conecciones de otros miembros de la red p2p
main() {
      master = socket.bind(SERVICE_IP, SERVICE_PORT)
      while true {
            client = master.accept()
            atender_solicitud(client)
            client.close()
      }
}
```

b) Usamos 2 mensajes: “solicitud” (contiene un nombre de archivo buscado) y “respuesta” (contiene la IP donde se encuentra el archivo)

```
// Esta es la función invocada para acceder al servicio.
find (file) {
      skt = connect(mi_ip)
      skt.send(construir_solicitud(file))
      respuesta=skt.read()
      skt.close()
      return  contiene_ip(repuesta)
}

//procesa mensajes que implementan el servicio
cuando atender_solicitud(client)
      solicitud = client:receive()
      file=parsear_solicitud(solicitud)
      if mi_archivo(file) {   //disponible localmente
            client.send(construir_respuesta(mi_ip()))
      } else {
            if (file<mis_archivos[1]) {   // orden lexicografico
                  skt = connect(prev_ip(ip))
            } else
                  skt = connect(next_ip(ip))
            }
            skt.send(solicitud)
            respuesta=skt.read()
            skt.close()
            client.send(respuesta)
      }
}

//servidor de sockets para recibir conecciones de otros miembros de la red p2p
main() {
      master = socket.bind(SERVICE_IP, SERVICE_PORT)
      while true {
            client = master:accept()
            atender_solicitud(client)
            client:close()
      }
}
```

Solución Problema 2

a)
i. El nodo B responderá con un mensaje ICMP echo reply.

ii. El Round Trip Time es el tiempo que tarda el paquete ICMP echo request en llegar a B más el tiempo que tarda el paquete ICMP echo reply en llegar a A. Los componentes del retardo son: Retardo de procesamiento (d_proc), Retardo de cola (d_queue), Retardo de transmisión (d_trans), Retardo de propagación (d_prop). Como A y B están directamente conectados (no hay nodos intermedios), solo se deben considerar los retardos generados por A y B. El RTT por lo tanto estará formado por d_procA debido al procesamiento en A del echo request, d_procB en B por echo reply, d_queueA y d_queueB por las colas en ambos nodos, d_transA y d_transB, el retardo de transmisión de los paquetes en ambos nodos y 2*d_prop, el retardo de propagación en el link entre A y B para el echo request y el echo reply.
Por lo tanto, RTT= d_procA+d_procB+d_transA+d_transB+d_queueA+d_queueB+2*d_prop

iii. Si los nodos tienen infinitos recursos y no hay congestión, entonces d_proc=0 y d_queue=0 para ambos nodos.

La nueva expresión de RTT será:
RTT=d_transA + d_transB + 2*d_prop.

Los retardos d_transA y d_transB son variables, depende del tamaño del paquete que se está transmitiendo. El retardo de propagación es constante ya que esta dado por la velocidad de propagación y la distancia entre A y B, la cual asumimos no puede variar. Como el echo reply es un paquete con el mismo tamaño que el echo request entonces d_transA = d_transB. Por lo tanto RTT(P)= 2*d_tans(P) + 2*d_prop.

El retardo de trasmisión para un paquete de tamaño P en un enlace con velocidad de trasmisión (throughput) v es igual a P/v, luego:

RTT(P)= 2*d_prop + (2/v)*P

Esta es una función lineal (una recta), donde P es el tamaño del paquete. Se representa por una recta donde la cosntante **2*d_prop** es el valor de la función para P=0 (el corte con el eje de las ordenadas), y **2/v** es la pendiente de la recta.

b)

i. Cada tamaño del paquete Pi dará valores RTTi, que son puntos de la recta en función de P. Para determinar esta recta se debe hacer una regresión lineal con los valores RTTi obtenidos para cada Pi. Una vez obtenida la recta, se puede despejar el valor de v de la pendiente:
   y=a.x+b
   (a es la pendiente, b es la constante, y es el RTT y x el tamaño del paquete)
   Luego b=2/v → **v=2/b**
ii. Debido a que en la realidad no existen nodos con recursos infinitos y hay congestión en la red. Como se mencionó anteriormente, hay componentes que dependen de la congestión de la red y del procesamiento de los paquetes en los nodos, lo cual varía con el tiempo. Entonces, enviando varios ICMP echo request con el mismo tamaño se logrará una mejor estimación para cada Pi; una buena estimación es tomar el mínimo, que tiene buena probabilidad de corresponder a la situación ideal sin retardos de procesamiento ni de cola.