Solución Primer Parcial – 27 de setiembre de 2019

(ref: prc20190927.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 (6 puntos)

Considere un paquete de longitud L que tiene su origen en el sistema terminal A y que viaja a través de tres enlaces consecutivos hasta un sistema terminal de destino B. Estos tres enlaces están conectados mediante dos dispositivos de conmutación de paquetes. Sean d_i, s_i y R_i la longitud, la velocidad de propagación y la velocidad de transmisión del enlace i, para $i = 1, 2, 3$.
Los dispositivos de conmutación de paquetes retardan cada paquete un tiempo d_{proc} .

Suponiendo que no se producen retardos de cola, ¿cuál es el retardo total terminal a terminal del paquete en función de d_i, s_i, R_i ($i = 1, 2, 3$) y L?

Solución:

El primer sistema terminal requiere L/R1 para transmitir el paquete al primer enlace, dicho paquete se propagará por el primer enlace en d1/s1; el primer conmutador de paquetes agrega un retardo de procesamiento de d_proc; luego de recibir el paquete completo el conmutador requiere L/R2 para transmitirlo al segundo enlace el que agregará d2/s2 para propagarlo. Del mismo modo actúa el segundo conmutador, obteniendo: d_terminal-terminal = L/R1 + d1/s1 + d_proc + L/R2 + d2/s2 + d_proc + L/R3 + d3/s3.

Pregunta 2 (6 puntos)

Comente brevemente los conceptos de control de congestión y control de flujo en TCP.

Solución:

El control de flujo es un servicio de adaptación de velocidades (adapta la velocidad a la que el emisor está transmitiendo frente a la velocidad a la que la aplicación receptora está leyendo) con el fin de eliminar la posibilidad de que el emisor desborde el buffer del receptor.

El control de congestión es un servicio para mejorar el funcionamiento general de la red, más que para el beneficio directo de los procesos que se comunican. 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 buffers. Si esta situación se

mantiene estos buffers crecerán consistentemente.

El mecanismo de control de congestión de TCP regula el proceso emisor (cliente o servidor) cuando la red está congestionada entre el emisor y el receptor. En otras palabras, los mecanismos de control de congestión de TCP evitan que cualquier conexión TCP inunde con una cantidad de tráfico excesiva los enlaces y routers existentes entre los hosts que están comunicándose.

Pregunta 3 (9 puntos)

Mediante un navegador se descargó por primera vez el archivo de texto http://europa.eu/hola.txt

Indique la secuencia de mensajes de Capa de Aplicación y de Capa de Transporte que se intercambian. Para cada mensaje especifique a qué capa pertenece, a qué protocolo, el tipo de mensaje, y las direcciones y puertos involucrados, según corresponda.

Solución:

La solución propuesta no considera aspectos particulares que se pueden presentar en algunas configuraciones o topologías específicas, sino que se focaliza en los aspectos fundamentales y generales de los protocolos involucrados de acuerdo a cómo se abordaron en el curso.

Si se intenta descargar por primera vez dicho archivo disponible en un servidor web, utilizando un navegador y por lo tanto el protocolo HTTP de Capa de Aplicación y, referenciando al sitio web por su nombre, lo primero que se debe hacer es una resolución de nombres, en particular, resolver cuál es la dirección IP asociada al nombre "europa.eu".
Por lo tanto, lo primero que viaja desde la computadora hacia el servidor de DNS que tiene configurado en la estación de trabajo, es un mensaje DNS (protocolo de Capa de Aplicación), en concreto, un *query* de "cuál es la dirección IP asociada al nombre europa.eu"? (tipo A y eventualmente otro mensaje query del tipo AAAA) con puerto origen "alto", mayor a 1023, por ejemplo 59457 y puerto destino 53, del servidor DNS; el mensaje viaja como carga útil de un mensaje UDP (protocolo de Capa de Transporte).
Sin considerar los detalles de cómo el servidor dispone de la respuesta (handshake en el cual no está involucrada la computadora), la misma es entregada mediante un mensaje DNS, en concreto, un mensaje *response* conteniendo "la dirección IP asociada al nombre europa.eu es D1.D2.D3.D4" (por ejemplo, 147.67.34.45); el mensaje viaja como carga útil de un mensaje UDP con puerto destino "alto" el mismo al del mensaje *query*, y para seguir con el ejemplo, 59457 y puerto origen 53.
Luego, se establece una conexión TCP (protocolo de Capa de Transporte) entre la computadora y el servidor web que realiza el hosting del sitio "europa.eu", la que queda identificada completa y unívocamente por las parejas (IPcomputadora, 60120) (147.67.34.45, puerto 80).
Nota: 60120 es un puerto origen "alto", mayor a 1023.
Para que se establezca la conexión TCP, se realiza el siguiente 3 Way Handshake:
- Segmento TCP desde computadora a servidor: con bandera SYN=1
- Segmento TCP desde servidor a computadora: con banderas SYN=1 y ACK=1
- Segmento TCP desde computadora a servidor: con bandera SYN=1

Luego de establecida la conexión TCP, se envía un mensaje referenciando al

método GET de HTTP para obtener el archivo "hola.txt", siendo un mensaje desde la computadora hacia el servidor.
A continuación, se recibe el contenido del archivo, mediante un mensaje de respuesta 200 OK, provenientes desde el servidor hacia la computadora.
Nota: se supone que no hay redirección HTTP.
Finalmente, concluida la transferencia, es de esperar que se cierre la conexión TCP, lo que ocurre, en su consideración más general, mediante cuatro mensajes:
- Segmento TCP desde computadora a servidor: con bandera FIN=1
- Segmento TCP desde servidor a computadora: con bandera ACK=1
- Segmento TCP desde servidor a computadora: con bandera FIN=1
- Segmento TCP desde computadora a servidor: con bandera ACK=1

Pregunta 4 (6 puntos)

Mencione y explique brevemente en que consisten las dos funciones fundamentales de la capa de red.

Solución:

Las dos funciones fundamentales de la capa de red son las de reenvío y enrutamiento. El reenvío (forwarding) refiere a cuando un paquete llega al enlace de entrada de un router, y éste tiene que pasar el paquete al enlace de salida apropiado. Mientras que enrutamiento (routing) refiere a que la capa de red tiene que determinar la ruta o camino que deben seguir los paquetes a medida que fluyen de un emisor a un receptor. Los algoritmos que calculan estas rutas se conocen como algoritmos de enrutamiento.

El reenvío hace referencia entonces a la acción local que realiza un router al transferir un paquete desde una interfaz de un enlace de entrada a una interfaz del enlace de salida apropiada. Y el enrutamiento, al proceso que realiza la red en conjunto para determinar las rutas terminal a terminal que los paquetes siguen desde el origen al destino.

Para que la capa de red cumpla su función (transportar paquetes desde un host emisor a un host receptor) estas funciones se relacionan mediante la tabla de forwarding. Cuando un router recibe mensajes del protocolo de enrutamiento utiliza esta información para configurar su tabla de reenvío, la cual es consultada por la función de reenvío para realizar su función.

Pregunta 5 (8 puntos)

a) Dibuje las FSM del emisor y el receptor para RDT 2.0.

b) Explique cuál es el fallo fundamental de este modelo, y cómo se puede resolver.

Solución:

a) Ver teórico FSM rdt 2.0.

b) El modelo 2.0 no tiene en cuenta la posibilidad de que el paquete ACK o NAK pueda estar corrompido. Si un paquete ACK o NAK está corrompido, el emisor no tiene forma de saber si el receptor ha recibido o no correctamente

el último fragmento de datos transmitido. Una solución sencilla a este nuevo problema (y que ha sido adoptada en prácticamente todos los protocolos de transferencia de datos existentes, incluido TCP) consiste en añadir un nuevo campo al paquete de datos, y hacer que el emisor numere sus paquetes de datos colocando un número de secuencia en este campo. Puesto que estamos suponiendo que disponemos de un canal que no pierde datos, los paquetes ACK y NAK no tienen que indicar el número de secuencia del paquete que están confirmando. El emisor sabe que un paquete ACK o NAK recibido (esté alterado o no) fue generado en respuesta a su paquete de datos transmitido más recientemente.

Problema 1 (15 puntos)

Se desea implementar una aplicación con una arquitectura cliente-servidor de intercambio de mensajes que cumpla con las siguientes características:

- El servidor espera conexiones TCP en todas sus interfaces en el puerto 8081.
- Los clientes envían dos tipos de mensajes:
  I. **ECHO**: *texto*\n
  II. **EXIT** \n
- Al recibir un mensaje de tipo **ECHO** el servidor debe contestar al cliente con *texto*.
- Al recibir un mensaje de tipo **EXIT** el servidor deberá contestarle "**CLOSE** *ip_cliente*" donde ip_cliente es la ip del cliente que envió el comando y cerrar la conexión con dicho cliente.

Se pide:

a) Implemente en un lenguaje de alto nivel, utilizando las primitivas de la API de sockets del curso, el programa que ejecuta el servidor.
b) Implemente en un lenguaje de alto nivel, utilizando las primitivas de la API de sockets del curso, el programa que ejecuta un cliente que envía los mensajes I y II.

Solución:

a)

```
void sevidor(){
    int master = socket.tcp();
    master.bind (*, 8081);
    serverSock = master.listen();
    while (true){
        clientSock, err = serverSock.accept(); //espero conexiones
        if (err)
            break;
         thread.create(atenderCliente, clientSock); //hilo de atención para
el cliente
    }
    serverSock.close();
}

void atenderCliente (clientSock){
    exit = false;
    while (!exit){
        command = "";
        repeat
            data, err = clientSock.receive(); //recibo el stream del cliente
            command += data;
           until ((err == 'closed') || (find(command, '\n')) //recibí el
comando completo
        if (err == 'closed' || data = ""){
            clientSock.close();
            return;
        }

    //Armo la respuesta dependiendo del comando
        if (find(command, 'ECHO')){
            stream = remove(command, "ECHO")
        }
        else {
            ip, port = clientSock.getPeer();
            stream = "CLOSE "+ip;
            exit = true;
        }

//envío la respuesta por la conexión
        repeat
            remain, err = clientSock.send(stream)
            if (err == 'closed')
                clientSock.close();
                return;
            stream = remain;
        until remain == ""
    }
    clientSock.close();
}
```

b)

```
void cliente(){
    int master = socket.tcp();
    client, err = master.connect(IP_SERVER, 8081)
    msg[1] = "ECHO texto a replicar ….\n"
    msg[2] = "EXIT \n"

    if !(client, err == nil, failure){
      for i=1 to i=2 {
         stream = msg[i]
         remain = ""
         repeat //envío el comando por la conexión
                     remain, err = client.send(stream)
                 if (err == 'closed')
                    client.close();
                     return;
                 stream = remain;
              until remain == ""

          repeat  //recibo la respuesta
                   data, err = client.receive();
               respuesta += data;
                 until ((err == 'closed') || (find(respuesta, '\n')) //recibí
la respuesta completa
      }
    }
    client.close();
}
```