Examen - 29 de julio de 2009 (ref: sirc0907.doc)

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.
•Utilice una caligrafía claramente legible.
•Comience cada ejercicio y cada pregunta en una hoja nueva.
•Sólo se contestarán dudas de letra. No se aceptarán dudas de ningún tipo durante los últimos 30 minutos del examen.
•El examen es individual y sin material.
•Es obligatorio responder correctamente al menos 15 puntos en las preguntas teóricas.
•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).
•Duración: 3 horas.

Preguntas Teóricas

Pregunta 1 (10 puntos)

a) En un protocolo de capa de transporte: ¿cuál es la diferencia entre Control de Flujo y Control de Congestión?
b) ¿Cómo hace TCP para detectar la existencia de congestión en la red?
c) ¿La pérdida de paquetes siempre implica congestión? Justifique su respuesta.

Respuestas

a) El control de flujo intenta que el emisor no desborde el buffer del receptor transmitiendo muy rápido demasiados datos. El control de congestión intenta evitar que se pierdan paquetes debido a que hay demasiadas fuentes enviando muy rápido demasiado tráfico a ser manejado por la red.

b) TCP asume que la perdida de paquetes (o grandes retardos) implican congestión.

c) No. La pérdida de paquetes tambien puede darse por una capa de enlace defectuosa, por ejemplo.

Pregunta 2 (8 puntos)

a) Describa los pasos del “Three Way Handshake” para el inicio de una conexión TCP.
b) Mencione dos casos (aplicaciones, servicios, etc.) en los que resulta mas apropiado usar UDP que TCP. Justifique su respuesta.

Respuestas

a)
Paso 1: el host cliente envía un segmento SYN al servidor - especifica el nro. de sec. inicial - sin datos

Paso 2: el host servidor recibe el SYN, responde con un segmento SYN-ACK sin datos
el servidor reserva buffers , especifica el nro.de sec. inicial del servidor

Paso 3: el cliente recibe el SYN- ACK, responde con un segmento ACK, que puede tener datos

b) Por ejemplo: streaming de video y telefonia sobre internet. Son casos en los que el control de congestión de TCP o el tiempo consumido por el establecimiento de conexion es contraproducente por tratarse de aplicaciones de tiempo real en las que mas vale perder algunos paquetes que atrazarlos todos.

Pregunta 3 (8 puntos)

a) ¿Cuál es la utilidad del protocolo ARP?
El protocolo ARP (Address Resolution Protocol) traduce entre la direcciones de red (dirección IP) de un equipo y su dirección de enlace MAC.

b) Describa su funcionamiento.
El procedimiento es el siguiente,
- *A quiere enviar un datagrama a B, y la dirección MAC de B no está en la tabla ARP de A.*
- *A realiza un broadcast de un paquete ARP query, conteniendo la dirección IP de B (Dirección MAC destino = FF-FF-FF-FF-FF-FF)*
- *Todas las máquinas en la LAN reciben el ARP query*
- *B recibe el paquete ARP y responde a A con su dirección MAC (ARP reply) . La trama es enviada a la dirección MAC de A (unicast)*
- *A salva (cache) el par direcciones IP-MAC en su tabla ARP hasta que la información se considere vieja (timeout)*

La información obtenida se almacena por un período de tiempo en la tabla ARP, donde consultará antes de realizar el procedimento para obtención de la dirección MAC.

Pregunta 4 (8 puntos)

a) Describa el mecanismo de “Reverse Path Forwarding” utilizado para la construcción de árboles de distribución en redes multicast.

Cuando un paquete multicast ingresa a un router, el router realiza un “RPF Check”, que consiste en verificar si la dirección IP de origen del paquete es alcanzable vía alguna de las rutas que salen por la interfaz de entrada del paquete.
Si el “RPF Check” se cumple, entonces el paquete se inunda (flooding) por todas las interfaces del router que tienen clientes del grupo de multicast destino del paqute. En caso contrario, se descarta.
Los routers que reciben el paquete y NO tienen clientes pueden enviar un mensaje PRUNE hacia arriba para no recibir mas paquetes.
Se puede ver que este mecanismo garantiza la creación de un árbol de distribución implícito (i.e. Nadie en la red tiene una representación del árbol, sin embargo el mismo existe) sin loops.

b) Describa los mensajes básicos del protocolo IGMP para el mantenimiento de grupos de multicast.

•MEMBERSHIP QUERY (o QUERY solamente, en las transparencias está así) – utilizado por routers para conocer y descubrir que hosts estan escuchando ciertos grupos de multicast
•MEMBERSHIP REPORT – utilizado por hosts para informar a los routers que lo escuchan que se está uniendo a un grupo de multicast
•LEAVE – Enviado por un host cuando este abandona un grupo de multicast

Pregunta 5 (6 puntos)

a) ¿Cuál es la diferencia entre la confidencialidad de un mensaje y la integridad de un mensaje?

Confidencialidad es la propiedad de que un mensaje en texto plano no puede ser determinado por un atacante que intercepte la versión cifrada de dicho mensaje.

La integridad de un mensaje es la propiedad de que el receptor del mismo pueda detectar cuando el mensaje enviado (cifrado o no) fue alterado en el camino. Se trata de conceptos (propiedades) distintas.

b) ¿Puede tener una de ellas sin la otra? Justifique su respuesta

Sí se puede tener una sin la otra. Un mensaje cifrado que es alterado durante su transmisión puede continuar siendo confidencial (el atacante no puede determinar el texto plano original) pero no tenemos integridad del mensaje si la alteración no es detectada en el destino.
De igual forma, un mensaje que es alterado (y detectado) en el tránsito entre el origen y el destino del mismo, puede ser enviado en texto plano y por lo tanto en ese caso no se tiene la propiedad de seguridad de confidencialidad.

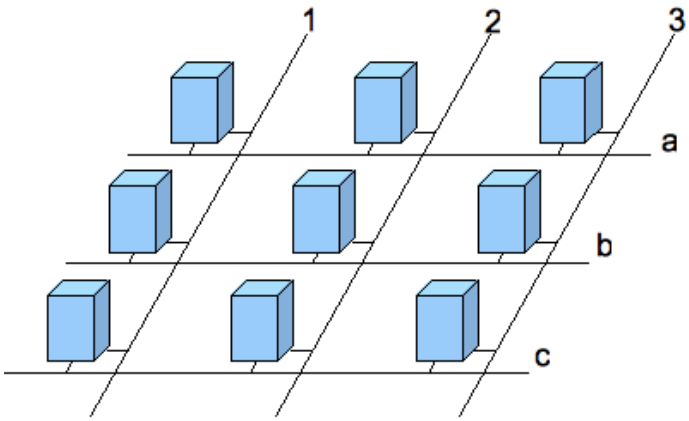
c) ¿Puede descifrar el hash de un mensaje para obtener el mensaje original? Justifique su respuesta.

No. Ésto se debe a que la función de hash es una función en un sólo sentido ("one-way").

Problemas Prácticos

Problema 1 (30 puntos)

Por razones de desempeño, un proyecto decidió eliminar los protocolos de capa de red y transporte, reemplazándolos por su propio protocolo, transmitiendo información entre nodos directamente a nivel de capa de enlace (p.e. Ethernet). De esta forma -dicen- pueden evitar el overhead generado por éstos, por aspectos no necesarios de los mismos.
Los equipos se disponen en una malla, con cada equipo conectado a dos dominios de colisiones, con dos tarjetas de red independientes. Sea la siguiente representación esquemática para un diseño plano de 3x3 nodos:



Cada nodo conoce las direcciones de sus adyacentes y las tiene almacenadas en una tabla, que es estática a lo largo del tiempo. Por ejemplo, el nodo **1a** conoce las direcciones físicas de los otros nodos en el dominio de colisiones **a**, así como, las direcciones de sus adyacentes en el dominio de colisiones **1**.
La tabla correspondiente al nodo **1a** sería:

```
interfaz nodo dirección
   a     2a   <dir_2a>
   a     3a   <dir_3a>
   1     1b   <dir_1b>
   1     1c   <dir_1c>
```

Notar que los nodos se identifican por su posición en la grilla, por ejemplo **3b**.
El algoritmo de ruteo que se desea implementar funciona de la siguiente manera:

- •si el nodo destino es él mismo, el mensaje se copia localmente
- •si el nodo destino es adyacente, transmitir directamente al destino.
- •si el nodo destino no es adyacente, transmitirlo a uno adyacente, indicándole a éste que lo re-envíe al correcto, que sí debe ser adyacente a él.
- •si un nodo recibe un paquete que debe re-transmitir y no es adyacente al destino, debe descartarlo silenciosamente.

Algunos ejemplos de ruteos válidos son:

```
1a 8  1a: copia local sin transmisión física
1a 8  3c: 1a 8  3a 8  3c
```

Un ejemplo de ruteo inválido sería:

```
1a 8  3c: 1a 8  2a 8  2b 8  2c 8  3c
```

De esta forma, todos los equipos están a un máximo de 2 hops de distancia uno de otros, siendo las decisiones de ruteo muy simples y eficientes. Las primitivas disponibles a nivel de capa de red son:

`int send(intr, dir, msg)`	transmite el mensaje msg a la dirección dir a través de la interfaz intr.
`int receive(interfaz, buffer)`	recibe un mensaje de cierta interfaz y un buffer, y devuelve el tamaño del mensaje almacenado en el buffer.

Se desea implementar un protocolo del tipo stop-and-go que resuelva este problema de ruteo, ofreciendo un servicio orientado a streams. Se pide:

Parte a)
i.Declare los prototipos de las funciones mínimas que este protocolo debe ofrecer a una aplicación que lo utilice, al estilo de la interfaz de sockets habitual.
ii.Proponga formato de mensajes y estructuras de datos para los mensajes necesarios para implementar un protocolo que nos permita conectar nodos de esta malla. ¿Es necesaria la presencia de un campo TTL (similar al presente en IPv4) en este protocolo? Justifique.

Parte b)Brinde una implementación detallada de alto nivel del mismo.

Solución

Parte a)
i.Declare los prototipos de las funciones mínimas que este protocolo debe ofrecer a una aplicación que lo utilice, al estilo de la interfaz de sockets habitual.

Si queremos dar una abstracción similar a la de sockets provista por TCP/IP, deberíamos proveer un constructor que nos permita identificar un extramo de la conexión:

`canal(u_int puerto) ;` // establece un puerto de atención (servidor)
`canal(addr direccion, u_int puerto)` // establece un canal con el destino espcif (clnt)

Las primitivas fundamentales que debe ofrecer son: accept, connect, close, read y write.

`canal.accept();` Coloca un canal recién definido a la espera de conexiones por parte de un cliente.
`canal.connect();` Conecta un canal recién definido a un servidor
`canal.close();` Finaliza la conexión activa
`buf canal.read();` Lee del canal devolviendo en buf el resultado
`canal.write(buf);` transmite buf a través del canal.

Todas las condiciones de error se informan a través de excepciones.

ii.Proponga formato de mensajes y estructuras de datos para los mensajes necesarios para implementar un protocolo que nos permita conectar nodos de esta malla. ¿Es necesaria la presencia de un campo TTL (similar al presente en IPv4) en este protocolo? Justifique.

Un conjunto de mensajes posibles serían:

```
class mensaje {
  boolean id = 0;
  addr dest;
  uint p_dest;
  addr orig;
  uint p_orig;
  boolean nro_seq;
  byte datos[MAX_LEN];
}

class ack{
   boolean id=1;
   nro_seq
}
```

Asumimos que tenemos la tabla de adyacentes disponible.

Un campo equivalente al TTL es innecesario, pues no hay mensajes que puedan ser forwardeados más de una vez. No existen rutas por defecto, por ejemplo, ni posibilidades de ciclos en el enrutamiento/forwarding.

Parte b)Brinde una implementación detallada de alto nivel del mismo.

Una implementación posible se puede estructurar con dos threads, cada uno atendiendo la llegada de paquetes por cada interfaz de red y las transmisiones se pueden controlar desde el thread que inicia la transmisión.

Thread de recepción atendiendo la interfaz x:

```
while (1) {
   int len = receive(int, buf);
   message m = new message(buf,len);
   if myself(m.dest) {
      canal c = abiertos.get(m.p_dest);
      if (c==null) {
           //no hago nada, nadie espera este mensaje
      } else {
        c.setMessage(m); //sea del tipo que sea ack o mensaje.
        c.notify(); //despierta al thread a la espera
      }
   } else { //el destino no es este nodo
      if adyacentes.isAdyacente(otra_interfaz(x),m.dest) {
         send(otra_interfaz(x), adyacentes.getDir(m.dest),m);
      } else {
         //descarto por no ser adyacente.
      }
   }
}
```

Se utiliza una colección **abiertos** que contiene todos los canales abiertos, así como, una colección **adyacentes** que contiene la tabla de adyacentes.

La rutina **accept()** simplemente registra un canal en la colección abiertos y establece el número de mensaje en espera en 0.

Al momento que un cliente crear un canal, se toman las decisiones de ruteo/forwarding y quedan establecidas de forma estática para toda la conexión.

La rutina connect() toma los parámetros introducidos en el constructor para definir la conexión y tendría la siguiente implementación:

```
if adyacentes.isAdyacente(direccion) {
   if = adyacentes.getInterfaz(direccion);
   addr = adyacentes.getDir(direccion);
} else {
   if = default; //p.e. La que me comunica por la columna
   addr = adyacentes.getDir(MY_ADDRESS.getCol()+direccion.getRow());
}
receive_seq=0;
send_seq=0;
```

Las variables if y addr son privadas de cada objeto canal, y son fijos a lo largo de la

El procesamiento central de la rutina **write()** se implementaría de la siguiente forma:

```
void write(byte[] buf) {
  int falta = buf.len;
  repeat {
    message m = new message();
    m.setDest(this.direccion,this.d_port);
    m.setOrig(MY_ADDRESS,s_port);
    m.setNroSeq(this.nro_seq);
    int largo = (falta<MAX_LEN ? falta : MAX_LEN);
    m.setDatos(buf[buf.len-falta, buf.len-falta+largo]);
    this.nro_seq != this.nro_seq; //ya cambio el número de secuencia
    repeat
      send(this.if, this.addr,m);
      wait(this,T_OUT);
    util (recibo_ack)
    this.send_seq=!this.send_seq;
    falta-=largo;
  } until (falta==0)
}
```

El procesamiento central de la rutina **read()** se implementaría de la siguiente forma:

```
byte[] read() {
  boolean recibido = false;
  repeat {
     if(this.mensaje==null){
       repeat {
          wait(this); //espero a que llegue un mensaje y me notifique
       until (this.mensaje!=null)
     }
     if (this.mensaje.getSequencia()==this.receive_seq) {
       this.receive_seq=!this.receive_seq;
       send(this.if, this.addr,new ack(this.send_seq));
       this.send_seq=!this.send_seq;
     }
  } until recibido;
  return message.getDatos()
}
```

La rutina **close()** des-registra el canal de la tabla de abiertos.

NOTAS:

No se hace el 3-way handshake pues al ser un protocolo no orientado a ventanas, podemos sincronizar el comienzo de la transmisión en 0 para ambos extremos, y pues tampoco nos preocupa el problema de seguridad de la predictibilidad del comienzo de la sesión.

El número de secuencia utilizado (0 o 1) se utiliza para evitar duplicados solamente.

La decisión de ruteo se hace teniendo en cuenta la tabla de adyacentes, que es conocida y estática a lo largo del tiempo. Cuando el destinatario de un mensaje NO es adyacente (la búsqueda en la tabla falla) se elije un nodo que si es adyacente de la tabla, y que es adyacente al nodo destino y a éste se le envía el mensaje, que lo re-envía al nodo destino en un paso solamente. Esto es posible por la estructura de la red.

Problema 2 (30 puntos)

El sitio *naivespamming* provee una aplicación web para detección de correo no deseado. La aplicación funciona por medio del protocolo HTTP, y se basa en una interacción sencilla que consta de un **`request`** y un **`response`**. Para saber si un correo es spam, se debe enviar un pedido HTTP a la url **`www.naivespamming.org`** pidiendo el objeto:**`isspam.cgi?sender=<IP_ADDRESS>&subject=<SUBJ>`** Donde se sustituye **`<SUBJ>`** por el asunto del correo e **`<IP_ADDRESS>`** por la dirección IP (en formato decimal separado por puntos) del servidor de correo indicado en la dirección del emisor (**`MAIL FROM`**). Este pedido genera dos posibles respuestas:

•Si la respuesta es el código de "no encontrado" (404) significa que *naivespamming* considera que el correo es no deseado.

•Si la respuesta se obtiene correctamente (200) significa que *naivespamming* considera que el correo es legítimo.

Se desea enriquecer un servidor de correo que recibe los mensajes mediante el protocolo SMTP, para que implemente un filtro de correo no deseado utilizando la aplicación de *naivespamming*. El siguiente pseudocódigo estilo java muestra el proceso que ejecuta un hilo de atención SMTP del servidor.

```
 1.void getMailFromClient(Socket s) {
 2.   bool end = false;
 3.   while (!end) {
 4.      try {
 5.         String clientName = "", from = "", to = "", data = "";
 6.         s.out.writeLine("220 smtp.myserver.com ESMTP Postfix");
 7.         String line = s.in.readLine();
 8.         if (line.startsWith("HELO")) {
 9.            clientName = parseHelo(line);
10.                    // obtiene el nombre del servidor de la linea HELO
11.               s.out.writeLine("250 Hello " + clientName + ", I am glad to meet
you");
12.         }
13.         else if (line.startsWith("MAIL FROM")) {
14.            from = parseFrom(line);
15.                    // obtiene el nombre del emisor de la linea MAIL FROM
16.            s.out.writeLine("250 OK");
17.         }
18.         else if (line.startsWith("RCPT TO")) {
19.            to = parseTo(line);
20.                    // obtiene el nombre del receptor de la linea RCPTO TO
21.            s.out.writeLine("250 OK");
22.         }
23.         else if (line.startsWith("DATA")) {
24.            s.out.writeLine("354 End data with <CR><LF>.<CR><LF>");
25.            line = s.in.readLine();
26.            while (line != ".") {
27.               data += smtpUnstuff(line); // remueve el byte stuffing de SMTP
28.               line = s.in.readLine();
29.            }
30.            saveToFile(from, to, data); // guarda el correo recibido en disco
31.            s.out.writeLine("250 OK mail saved");
32.         }
33.         else if (line.startsWith("QUIT")) {
34.            end = true;
35.            s.out.writeLine("221 Bye");
36.         }
37.         else {
38.            s.out.writeLine("500 Command unrecognized");
39.         }
40.      } catch (TimeoutException exc) {
41.         end = true;
42.         s.out.writeLine("554 Time out");
43.      }
44.   }
45.   s.close();
46.}
```

Para implementar el filtro de correo no deseado, se modificará el método **`getMailFromClient`** haciendo que invoque la aplicación web de *naivespamming* antes de guardar el mail en un archivo. Solamente lo guardará si *naivespamming* considera que no es spam.

Se pide:

Parte a)
Implementar la invocación al DNS necesaria para obtener la IP del servidor de correo dado su nombre de dominio. Esto debe realizarse en una función en el cabezal:

`string getMailServer(string hostName)`

que devuelva la dirección IP en su formato decimal separado por puntos. Se debe realizar la invocación vía UDP, la IP del servidor es **`DNS_SERVER_IP`** y el puerto es **`DNS_SERVER_PORT`**. Para este punto ya tiene implementada las siguientes funciones:

`string createDNSMessage(string regName, string regType)`
que crea un mensaje DNS preguntando por el tipo de registro **`regType`** con el nombre **`regName`**.

`string getDNSValue(string dnsMessage, string regType)`
que toma el mensaje DNS (por ejemplo la respuesta obtenida) y devuelve el valor asociado al registro indicado.

Solución:

```
string getMailServer(string hostName) {
   DatagramSocket s = new DatagramSocket();
   string dnsMessage = createDNSMessage(hostname, "MX");
   DatagramPacket d = new DatagramPacket(DNS_SERVER_IP, DNS_SERVER_PORT,
dnsMessage);
   s.send(d);
   try {
      DatagramPacket r = s.receive();
   } catch (TimeOutException exc) {
      return null;
   }
   return getDNSValue(r.getData(), "MX");
}
```

Parte b)
Implementar la función que invoca a *naivespamming* y devuelve el resultado. La misma debe tener el siguiente cabezal:

`bool isSpam(string ipAddress, string subject)`

Devuelve **`true`** si *naivespamming* considera que el correo es spam, y **`false`** en caso contrario. Puede asumir que se cuenta con una función **`queryStringStuff`** que realiza el stuffing de un string cualquiera para que pueda pasarse como valor en un **`query string`**.

Solución:

```
bool isSpam(string ipAddress, string subject) {
   try {
         Socket s = new Socket("www.naivespamming.org", 80);
         if (!s.connect()) {
            return false; // naivespamming no contesta, asumimos que no es spam
         }
         s.out.write(
              "GET isspam.cgi?sender=" + ipAddress +
              "&subject=" + queryStringStuff(subject) +
              " HTTP/1.1\r\n\r\n");
         string response = s.in.readLine();
         s.close();
         string[] result = response.split();
              // separamos la respuesta HTTP/1.1 <CODIGO> <VALOR>
         return result[1] == "200";
              // nos quedamos con la segunda palabra: el código de respuesta
   } catch (TimeOutException exc) {
      return false; // naivespamming no contesta, asumimos que no es spam
   }
}
```

Parte c)
Modificar el código de la función **`getMailFromClient`** para que realice la invocación de **`isSpam`** donde corresponda y con los datos apropiados, y guarde el correo en un archivo solamente si no es spam.
Solución:
Cambiar las líneas de la 22 a la 31 del pseudo de SMTP por las siguientes:

```
else if (line.startsWith("DATA")) {
   s.out.writeLine("354 End data with <CR><LF>.<CR><LF>");
   line = s.in.readLine();
   string serverName = "", subject = "";
   bool processingHeader = true;
   while (line != ".") {
      if (processingHeader && line.startsWith("Subject:" ) {
         string subject = line.substring(9);
            // eliminamos la expresión "Subject: "
      }
      else if (processingHeader && line == "") {
         processingHeader = false;
      }
      data += smtpUnstuff(line); // remueve el byte stuffing de SMTP
      line = s.in.readLine();
   }
   string serverName = from.split("@")[1];
      // nos quedamos con lo que está después de la "@"
   string serverIp = getMailServer(serverName);
   if (serverIp != null && !isSpam(serverIp, subject)) {
      // en el chequeo se agrega la condicion de que serverIp no sea null que es
el valor
      // de salida que nos da getMailServer cuando no pudo encontrar la
respuesta
      saveToFile(from, to, data); // guarda el correo recibido en disco
   }
   s.out.writeLine("250 OK mail saved");
}
```